情報処理学会論文誌　Vol. 52　No. 3　1365–1394 (Mar. 2011)

ソフトウェアの問題記述のための実行可能な形式化自然語の提案

日　野　克　重[†1]　　本位田　真一[†1,†2]

現場技術者が使用することを想定したソフトウェアの問題記述のための実行可能な形式化自然語（FNL；Formalized Natural Language）を提案する．ソフトウェア開発の多くの困難は，もとの問題（要求）とそれを実現するソフトウェアの記述との間の意味論的隔たりからきていると考えられる．もし，原問題に近い表現で問題を書くことができ，かつ，それが実行もできるようになれば，ソフトウェア開発過程が大きく改善されることが期待できる．これに近いねらいを持つ言語は形式仕様言語または要求記述言語と称され，すでに，Z，VDM，OBJ，および CSP など多数提案されているが，現時点ではそれらが産業界に十分普及しているとはいい難い．その理由としては，（ｉ）読み書きが難解である，（ii）検証中心であり実行可能性が弱い，（iii）問題世界の統合的記述性が十分でない，（iv）本質的表現力の問題などの困難が指摘されている．しかし，これまで提案されている言語の延長上でこれらの困難を打開できるシナリオ像は必ずしも明確ではないように見える．本稿ではあえて出発点に戻り，これらの困難が最初から存在しない言語を提案する．

A Proposal for Software Specification by an Executable and Formalized Natural Language

Katsushige Hino[†1] and Shinichi Honiden[†1,†2]

We propose a formalized natural language (FNL) for software specification, which is executable on computers. Almost all the difficulties in software development seem to come from the semantic gaps between original problems (or requirements) and software programs. Thus, we can expect to radically improve the software development process, if it becomes possible to immediately execute the problems described in formalized expressions which are close to the original ones of problems. Many languages with similar aims have been proposed as formal specification languages including Z, VDM, OBJ, and so on. However, these have not penetrated broadly enough across the industry at present. The reasons for this are as follows: (1) difficult to read and write; (2) verification oriented but weak at executability; (3) not integrated enough to describe static structure, dynamic behavior and concurrency of the problem world; (4) weak with the essential expressiveness. A clear scenario to break through these difficulties may not be found based on the extension of the existing languages. Therefore, we propose a new language (FNL) which eliminates the above issues, daringly returning to the origin of software specification.

1. はじめに

現場技術者が使用することを想定したソフトウェアの問題記述のための実行可能な形式化自然語（FNL; Formalized Natural Language）を提案する．

ソフトウェア開発の生産性および品質の向上にとって，ソフトウェア記述言語はいうまでもなく最も基礎的な役割を果たす．ソフトウェアの開発とは，つきつめるところ，もとの問題（要求）とそれを実現するソフトウェアの記述との間の表現上の写像操作であり，ソフトウェア開発のあらゆる困難は，これら両者の間の（垂直方向の）意味論的隔たりからきていると考えられるからである．もし，原問題に近い表現で問題を書くことができ，かつ，それがそのまま実行もできるようになれば，設計からプログラミングならびにテスト工程が大幅に省略され，ソフトウェア開発過程が大きく改善されることが期待できる．

これに近いねらいを持つ言語は形式仕様言語あるいは要求記述言語と称され，すでに，Z[1)]，VDM[2)]，OBJ[3)]，および CSP[4)] などをはじめとして多数提案されている．その中には実用になっているものも多いが，現時点ではそれらの言語が産業界に十分普及しているとはいい難い．その理由としては，以下の困難が指摘されることが多い[5)–8)]：（ｉ）読み書きが難解である（数学的すぎる），（ii）検証中心であり実行可能性が弱い，（iii）問題世界の統合的記述性が十分でない（たとえば，静的構造，動的振舞い，および並行性を統合的に記述できない），（iv）本質的表現力の問題（たとえば，状態モデルで並行システムを記述しきれるか，代数的記述はプログラミングと違いがないように見えるなど）．しかし，これまで提案されている言語の延長上でこれらの困難を打開できるシナリオ像はかならずしも明確ではないように見える．

本稿では，あえて出発点に戻り，上述の困難が最初から存在しない言語を提案する．2 章

†1 東京大学大学院情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo
†2 国立情報学研究所
National Institute of Informatics

© 2011 Information Processing Society of Japan

で言語仕様の概要を述べ，3 章で典型的な記述例を示す．4 章ではその実行方式を示し，5 章でこの言語について評価・議論する．6 章はむすびである．

2. 言語 FNL の概要

2.1 設計方針

言語 FNL は，ソフトウェア開発現場の技術者が使うことを想定する．具体的には，以下の要請を課し，それらを最大限に満足する言語仕様を追究する．

＜要請 1＞記述性

現場技術者（一般人）の認知心理的要素を考慮して，記述性の要請をさらに以下のように具体化する．

＜要請 1a＞**自然語指向**[9),10)]：問題を記述する言語は自然語に近いことが望ましい．自然語を指向することにより以下のような効果が期待できる．

- 原初の問題や要求は人が（非形式的な）自然語や図を使って表現するので，問題記述も自然語記述に近いほうが原問題と問題記述との一致性を確認しやすい．
- 有用なソフトウェア概念も元来自然語の中に含まれていたものとすれば（例：継承概念や多相概念など），自然語を指向することにより一般技術者にとってもそれらの概念を自然なものとして享受しやすくなる．

ただし自然語そのものでは，明らかに後述の要請 2 形式性&厳格性をみたせない．

＜要請 1b＞**推論性**

記述から読み取れる内容はすべて読み取ってくれる言語が望ましい．開発者の思考力が増幅され問題記述を楽しめるようにしたい．

＜要請 1c＞**存在論**[11)–14)]：言語の背後にある存在論が明示されていること．ソフトウェアの記述言語は世界の存在論をより反映する方向に発展しているとすれば，言語発案時にその存在論を明示することは今後ますます重要性を増すものと思われる．

＜要請 1d＞**統合性**：世界の静的構造，動的振舞い，並行性が統合的に記述できること[12)]．現場技術者が扱うソフトウェアシステムのほとんどはこれら 3 つの要素が密に関係しあって現れる．しかし，それらを有機的に統合した言語は少ない．

＜要請 1e＞**素朴&直接性**：対象を直接指示する言語要素だけがあること．オブジェクト指向言語における「class」をはじめとして，「precondition」，「postcondition」，「atomically」，さらには型宣言部など，ソフトウェア記述言語には問題記述をさらに上から説明する多数のメタ的言語要素が導入されることが多い．しかし FNL が形式言語として暗に模範としている数学言語や述語論理にはこうしたメタ的言語要素はない．

＜要請 1f＞**問題純粋性（実現隠蔽性）**：問題と実現とを鋭く区別し，問題記述に集中すること．元の問題に現れない実現概念は問題記述には現れないこと．

たとえば，入力&出力やスレッド，さらには排他制御，処理の順序性（ex. FIFO），および内部的状態などの概念は（通常は）問題概念ではなく実現概念であるので問題記述からは排除したい．

＜要請 2＞**形式性&厳格性**：形式的意味論が与えられていること．言語のうえで厳密な思考計算ができること．形式意味論は，「記述」と「実行」との独立性を保証し，かつ両者を結合する役割を果たす．

＜要請 3＞**実行可能性**：プログラムの自動導出などを含めて実行可能であること．あるいはその具体的な見通しがあること．これは単なる「記述」言語では満足しない現場技術者にとって重要な要件である．

2.2 言語 FNL の存在論

図 1 に言語 FNL の存在論を示す．これはいわば「言語 FNL では世界をどのように把握しどのように記述するか」を宣言するものである[⋆1,⋆2]．

図 1 の FNL の存在論が含意することを補足する．

まず，全体として，この存在論では実現概念（プログラミング概念）は完全に排除されている．「実体と関係」は，世界の静的構造の抽象化に相当する[⋆3]．「行為と変化」は，世界の動性の導入である．既存の時間論理も動的世界を表現するが，世界の変化が何によってもたらされるかについては言及しない．本存在論では，世界の変化は，なんらかの主体による行為によってもたらされるといっている．行為は時区間論理に通じるものである．「行為主体」は，プロセス（あるいはスレッド）とそれが実行するプログラムを一体のものとして統合し抽象化したものであり，並行性の把捉を意識している．「世界外存在」は，開放型システムを意識したものである．「創成者」はシステム記述者，「使者」はシステムに対するコマンド操作やメッセージなどを一般化・抽象化したものである．「論理」は，本存在論が

⋆1 本稿では「存在論」という用語は，単なる「語彙体系」という意味ではなく「問題世界を把握するときの基本枠組み」という意味で用いる．より形式的には，（元来同語反復しか生まない）論理という形式に対して付加すべき内容（語彙や公理系など）という意味で用いる．

⋆2 図 1 に示した存在論が現場技術者には難解であると評するのは妥当ではない．その表現と実質内容とを区別すれば，ここに示した存在論の内容自体は，自然語を解する一般人なら暗黙に了解していることであろう．自然語の読み書き能力とその文法理論の理解とは別のことであるが，それと類似した関係にある．

⋆3「実在性」は Free Logic[54)] でいう existence とほぼ同じ概念である．述語論理の存在概念より強い．

© 2011 Information Processing Society of Japan

①実体と関係：世界の中にはいくつかの実体が存在し，それらの間には種々の関係が成り立っている．実体とは，種が付与された個体のことである．実体は，実在性をもっている．実体はまた，いくつかの属性をもちえる．なお，成立（非成立）している関係のことを事態とよぶ．世界の状態とは，事態の集まりのことである．
②行為と変化：世界の状態は時間とともに変化する．そしてその変化は行為によってもたらされる．行為とは，世界の状態を参照しながら新たに事態を成立（非成立）させることである．行為には脈絡がある．
③行為主体：行為にはかならずその行為の主体が唯一存在する．行為主体になり得るのは，実体と外部使者（後述）だけである．行為主体たちは，それが実在しているかぎり，その行動本性に従って，世界をながめながら，おのおの独立に行動する．
④世界外存在：世界を創成するのは創成者である．創成者は世界の外の存在である．その意思は世界外使者（以下単に使者と呼ぶ）を介して世界に伝えられる．使者は行動するが，実在しない．
⑤論理：世界はつねに論理にしたがう．

図 1　言語 FNL の存在論
Fig. 1　Ontology of FNL.

述語論理（さらにはそれを拡張した内包論理）の存在論に依拠したものであることをいっている．

2.3　言 語 仕 様

(1)　言語仕様の骨子

FNL の言語仕様の骨子は以下のように要約される．

- FNL は，内包論理[15),16)] ⋆1に準拠し，その上に FNL 固有の存在論を反映した多主体並行型の時区間述語論理言語である．
- FNL における問題記述とは，以下の 4 種類の定義文の並び（順不同）である．
  - ①　各主体の行動本性（行動目標）
    - ①-a　各実体の行動本性
    - ①-b　世界（システム）外からの操作（これも外部使者という主体の行動本性と見なすことができる）
  - ②　問題全体に課せられる恒常的制約条件
  - ③　派生的語彙（記述者が導入した語彙で，他の語彙から定義されるもの）

⋆1 内包論理は，技術的には 2 ソート（もの ＋ とき）の型付き λ 論理と同一視できる[15)]．このように内包論理は論理体系として強力であるとともに存在論については無色であることが，FNL が準拠する論理体系として採用した理由である．

表 1　言語 FNL の基本詞
Table 1　Primitive words of FNL.

品詞	基本詞
定義詞	==
叙述詞	is　are　=
	becomes　become　:=
	does　do
普通名詞	Entity　Number
	String　Set
	（任意）
実在詞	existent　nil
属性詞	（任意）
関係詞	（任意）
事態詞	true　false
	（任意）
行為詞	（任意）

品詞	基本詞
論理詞	not　and　or
	if　->
限定詞	all　any　every
	some　many　the
主体詞	@
様相詞	=>　while　when
修飾詞	where　who
	which　whose
助詞	:　's　for

その他，算術詞，数詞，固有名詞，指示詞など．

（任意）：問題ごとに記述者が導入可能

すなわち，FNL では問題を「各主体の行動本性（目標）＋ 恒常的制約条件」の形式で記述する．派生的語彙の定義は，記述の階層化のためのものである．

- 実際には，FNL における文は，内包論理における正規の記法に構文糖衣を施した平易な自然語（英語）文の形で記述する．ただし，自然語には存在する複数形，語尾変化，副詞表現などは，現時点では意図的に導入していない．

以下，形式的詳細は付録 A1 にゆずり，FNL の言語仕様の要点を例文を使って示す．

(2)　構文規則および意味規則

(a) 基本詞

FNL の基本詞としては，**表 1** にあげるものがある．

なお，普通名詞は「型」の役割を果たす．当言語の基本型としては，

実体，自然数，文字列，使者，事態，行為

があり，普通名詞の Entity は実体，Number は自然数，String は文字列にそれぞれ対応する（使者，事態，および行為に対応する普通名詞はない）．記述者が導入する普通名詞はこれらの下位の型を示すことになる．

(b) 定義詞（==）

（例文）P == Q.

「P とは Q ということである」の意．ここで P は被定義項であり Q は定義項である．

P の中には，定義される語彙（派生的語彙）がちょうど 1 つだけなければならない．

© 2011 Information Processing Society of Japan

(c) 叙述詞，普通名詞，1 項関係詞

（例文）the Flower:f is red

「花 f は赤い」の意．叙述詞 is は，この文が現時点での事態を表現していることを示す．Flower と red はそれぞれ記述者が導入した普通名詞と 1 項関係詞である．f は指示詞である．なお文脈から一意に定まる場合は，普通名詞と指示詞のどちらか一方は省略可能である．本例文は，内包論理記法での「$\mathrm{red}(\mathrm{f}_{\mathrm{Flower}})$」の構文糖衣表現である．

（例文）the Flower:f becomes red

「花 f が赤くなる」の意．叙述詞 becomes は，この文が次時点で成立すべき事態（あるいは次時点でその事態を成立させる行為）であることを示す．本例文は，時相演算子（次時点詞）◎を導入した内包論理記法「$◎\ \{\mathrm{red}(\mathrm{f}_{\mathrm{Flower}})\}$」の構文糖衣表現である．なお，FNL の意味論には以下が含まれている．

- 「becomes (p)」（「◎ p」）は，単位時区間で実行される．ここで p は任意の事態文とする．

なお，以降，単位時区間で実行される行為を単位時区間行為とよぶ．

(d) 実在詞（existent，nil）

（例文）the Flower:f is existent

「花 f は実在している」の意．

（例文）the Flower:f becomes existent

「花 f が（次時点で）実在するようになる」の意．

（例文）the Flower:f becomes nil

「花 f が（次時点で）消滅する」の意．なお，実在性[54] に関する FNL の意味論には以下が含まれている．

- システム起動の初期にはどんな実体も実在していない．
- 実在していないものはどんな属性や関係も持たない．

(e) 助詞 's と属性詞

（例文）the Person:p's Age = the Number:24

「人 p の年齢は 24 歳である」の意．この文を記述したとき，Age が属性詞であること，ならびにその型が $<\mathrm{Person}, \mathrm{Number}>$ であること（すなわち Age は Person から Number への関数であること）を同時に定義したことになる．このように FNL では型推論[21] を行うので推論可能な型の明示的記述は省略することができる．

(f) 2 項関係詞

（例文）the Flower:f is on the Desk:d

「花 f は机 d の上にある」の意．本例文は，内包論理記法での「$\mathrm{on}(\mathrm{f}_{\mathrm{Flower}}, \mathrm{d}_{\mathrm{Desk}})$」の構文糖衣表現である．

（例文）the Flower:f becomes on the Desk:d

「花 f は（次時点で）机 d の上にあるようになる（すなわち机 d の上に置かれる）」の意．内包論理記法での「$◎\ \{\mathrm{on}(\mathrm{f}_{\mathrm{Flower}}, \mathrm{d}_{\mathrm{Desk}})\}$」の構文糖衣表現である．

(g) 事態詞（true，false）

FNL に組み込みの事態詞としては true および false がある．これらはそれぞれ通常の意味を持つ．そのほか，記述者が任意の事態詞を導入することができる．

(h) 行為詞

（例文）the Flower:f does grow

「花 f が成長する」の意．叙述詞 does の直後の grow は行為詞と解釈される．

なお，FNL の意味論には以下が含まれている．

- 基礎行為は，単位時区間で実行される．
  ここで，基礎行為とは他の行為によって派生的に定義されているのではない行為のこと．

(i) 論理詞（not，and，or，if）

通常と同じ意味を持つ．なお FNL では，これらの論理詞は事態文だけでなく行為文どうしを結合することができる．そのとき，「行為文 and 行為文」の「and」は同時実行の意味を持ち，「行為文 or 行為文」の「or」は選択実行の意味を持つ．なお or による選択実行では or 結合での記述の順の優先度で選択される．

(j) 限定詞（all，some，many，the）

（例文）for all Flower:f (f is on the Desk:d)
　　all Flower are on the Desk:d

これらの文は，内包論理記法における $\forall \mathrm{f}_{\mathrm{Flower}}\ \{\mathrm{on}(\mathrm{f}, \mathrm{d}_{\mathrm{Desk}})\}$ の構文糖衣表現である．

（例文）for some Flower:f (f is on the Desk:d)
　　some Flower:f is on the Desk:d

これらの文は，内包論理記法における $\exists \mathrm{f}_{\mathrm{Flower}}\ \{\mathrm{on}(\mathrm{f}, \mathrm{d}_{\mathrm{Desk}})\}$ の構文糖衣表現である．

（例文）for many(2) Flower:f (f is on the Desk:d)
　　many(2) Flower are on the Desk:d

「2 つ以上の花が机 d の上にある」の意．

© 2011 Information Processing Society of Japan

（例文）all Flower are on some Desk

「すべての花はなんらかの机の上にある」の意．内包論理記法における $\forall f_{Flower}\,\exists d_{Desk}\,\{on(f,d)\}$ の構文糖衣表現である．なお，限定詞が多重になった場合，限定順序は限定詞の左からの出現順序と同じである．

（例文）for all Flower:f (f becomes on the Desk:d)

「すべての花が机 d の上に置かれる」の意．このとき，すべての花が机の上に置かれるのは同時である．

(k) 主体詞（@）

（例文）@ Signal:s == while true {s becomes blue => s becomes yellow => s becomes red}

「信号 s とは永久に青，黄，赤になることを繰り返すものである（信号 s の行動本性）」の意．なお，主体性に関する FNL の意味論には以下が含まれている．

- 実体はそれが実在している限り，その行動本性に従って行動する．
- 他主体が成立させた事態を，それと同等以下の意思優先度の主体が恒常的制約条件を順守するために不成立にすることは許容されない．

(l) 時相詞（=>，while，when）

（例文）Q => R

逐次行為．すなわち「Q を実行して，その後 R を実行する」の意．なお，FNL の意味論には以下が含まれている．

- Q の実行と R の実行の間に他の主体の行為が割り込む可能性がある．

（例文）while P Q

継続行為．すなわち，「P が真である間，行為 Q を継続する．P が偽になれば行為 Q をやめる」の意．

（例文）when P Q

待機行為．すなわち「P が真になったとき，行為 Q を行う」の意．

(m) 修飾詞（where，which，whose など）

（例文）for all Flower:f where (f is on the Desk:d); {the Flower is red}

「机 d の上にあるすべての花は赤い」の意．なお，一般に FNL における「A:x where P」は，内包論理記法における「$\lambda x_A.\,P$」と同じ意味である．

（例文）all Flower which are on the Desk:d become red

「机 d の上にあるすべての花が（次時点で）赤くなる」の意．

(n) 算術詞（#，sum など）

（例文）# (Flower:f where (f is on the Desk:d))

「机 d の上にある花の個数」の意．

（例文）sum (Flower:f's Height where (f is on the Desk:d))

「机 d の上にある花の高さの総計」の意．

そのほか +，*，=，< などの算術演算子や数字は通常の意味を持つ．

(o) 恒常的制約条件（true == P）

（例文）true == for all Fork:f {not(many(2) Philo are using f)}

「true == P」（P は事態文も含む単位時区間行為文）と記述することにより，P が恒常的に成立する制約条件であることを定義することができる．これは内包論理記法での「A□P」に相当する（ここで「A」は後述の経路演算子であり，「すべての経路について」の意である）．ただし，この恒常的制約条件 P は単位時区間行為文でなければならず，その中に，時相詞（=>，while，when）および主体詞 @ は含まれていてはならない．

(3) 語用規則（仕様，操作，実行，現実的モデル）

FNL においては，（システムの）仕様，操作，および実行ということをそれぞれ以下のように形式的に定義する．

① 仕様：仕様とは，次の形の型 t（事態の型）の整合式である．

$(A_u == B_u)$ and $\cdots$ and $(C_v == D_v)$

ここで，u および v は型を表す．

② システムの起動：システムの起動とは，システムの仕様を真にすることである．

③ システムの操作：操作コマンド A_p（p は行為の型）を投入するとは，仕様内に定義されている世界外操作 A_p を真にすることである．

④ システムの実行：システムの実行とは，1 つの仕様と複数の操作によって課せられる制約を満足するモデルのうちの「現実的モデル」（後述）を構成することである．

⑤ 現実的モデル：主体たちの実行系列（いわゆる実行スケジュール；FNL 意味論における行為主体関数がそれに対応する；またそれは CTL [31] における経路とも同一視できる）ごとにそれぞれ複数のモデルがありうるが，それら各実行系列でのモデルのうち，「現実的な変化」を継続しているモデルのこと．現実的な変化とは以下の 2 つの条件をみたした変化のことである．

（i）極小変化性：変化が極小であること（モデル構成上必要でない変化は起きないこと．「極小」となっているのは，変化の非決定性を考慮したものである）．

© 2011 Information Processing Society of Japan

（ii）許容変化性：各主体の行為が起こす変化が，その主体の属性（主体の意思の優先度や主体の計画性など）に照らして許容される変化であること（たとえば，他主体が成立させた事態を，それと同等以下の意思優先度の主体が恒常的制約条件を順守するために不成立にすることは許容されない）．<注> 極小変化性および許容変化性の意味については，3 章記述例（例 3）で補足する．

ここで変化とは，時点 i から時点 i + 1 への推移における基礎定項たち（ただし行為定項を除く）の意味変化のことであり，その大小関係は基本的には集合としての包含関係で定義される．ただし，基礎定項 existent の意味変化は無限大と評価するなどの考慮がされる．現実的モデルの直感的理解は容易であるが，その厳密な定式化には非単調論理[18),19)] をさらに拡張する必要がある．それは今後の課題である．

3. 記　述　例

FNL による典型的な記述例を 5 題示す[*1]．

（例 1）**住所録問題**[20)]：住所録に名前と住所を登録する．また，名前から住所を検索する．

［**図 2** の FNL 記述の要点］

- add と find という住所録世界の外からの 2 つの操作を定義している．システム操作者がこれらの操作を指示すると，「使者」が，定義されている操作内容を実行する（ここで述べた操作と使者の関係は他の記述例でも同様である）．
- 多くの形式仕様言語に存在する型定義部が FNL 記述にはない．それらは FNL が型推論[17),21)] により決定する．本例題の場合も，add，Address，および find の型を明示的には記述していないが，明示的に指定された p_{Person}，a_{String}，および，論理定項「==」（内包論理記法[15)] では「≡」）の型が <X<Xt>> であることをもとに順次，型推論することにより，それらの型を決定する．
- Z 記述などで必要になる初期化記述（システム初期には Address は空集合であることなどの記述）は FNL 記述にはない．それは言語 FNL の意味論にすでに含まれていることなので（付録 A1 参照），問題記述者はそれについて記述する必要はない．

（例 2）**食事する哲学者**[22)]：哲学者たちが食事と思考を繰り返す．食事のときは両脇のフォークを使う．1 つのフォークを複数人が同時に使用することはできない．

*1 本章の 5 つの記述例ではいずれも，問題環境の初期設定（哲学者の生成および配置や迷路の生成および設定など）に関する記述は省略している．

```
 1:  add ( the Person:p, the String:a ) ==
 2:    p becomes existent  and
 3:    p's Address becomes a .
 4:  find ( the Person:p ) ==
 5:    does list p's Address .
```

図 2　住所録問題の FNL 記述
Fig. 2　Address book problem written in FNL.

```
 1:  @ Philo:p ==
 2:    while true {
 3:      while  ( p is not hungry )
 4:         { p does think } and
 5:      while  ( p is hungry )
 6:         { p does eat  } } .
 7:  the Entity:e is near the Entity:d ==
 8:    ( e is by d ) or ( d is by e ) .
 9:  true ==
10:    for all Philo:p, for all Fork:f;
11:      ( not ( many(2) Philo does use f ) ) and
12:      ( p does eat -> p does use many(2) Fork ) and
13:      ( p does not use any Fork which is not near p ) .
```

図 3　食事する哲学者問題の FNL 記述
Fig. 3　Dining philosophers problem written in FNL.

［**図 3** の FNL 記述の要点］

- 行 1〜6 は哲学者という主体の行動本性の定義であり，行 7〜8 は派生的語彙 near の定義である．行 9〜13 は本問題の恒常的制約条件（ex. 哲学者が食事するためには 2 本のフォークを必要とするなど）の定義である．
- これら恒常的制約条件はそれ自体，静的な記述であるが，哲学者の行 4 や行 6 の行為に影響する（たとえば，近くにあるフォークが他の哲学者によって使用されているときには食事できないので行 6 の実行は待つなど）．
- 同時実行制御（排他制御などを含む）について，問題記述の中では直接にはなんら記述していない（記述する必要がない）．FNL 意味論における「単位時区間行為」の概念はデータベース分野におけるトランザクションの概念を含んでいるので，図 3 の記述がすでにトランザクションを定義していることになる（これについては 5.1.2 項で詳しく議論する）．

（例 3）**スイッチ問題**[23)]：スイッチをオンにしたりオフにしたりする．ただし，どの時点でも，オンであるスイッチの個数は 1 個を超えない．

© 2011 Information Processing Society of Japan

```
1:  TurnOn ( the Switch:s ) ==
2:     ( s becomes on )  or
3:     ( s does not anything ) .
4:  TurnOff ( the Switch:s )  ==
5:     ( s becomes off ) .
6:  the Switch:x is off ==
7:     the Switch:x is not on .
8:  true ==
9:     not ( many(2) Switch are on ) .
```

図 4　スイッチ問題の FNL 記述
Fig. 4　Switch problem written in FNL.

［**図 4** の FNL 記述の要点］

- 行 1〜5 はスイッチの世界に対する外からの操作 TurnOn および TurnOff の定義であり，行 6〜7 は派生的語彙 off の定義である．本問題の制約条件（オンであるスイッチはたかだか 1 つであること）は，恒常的制約条件として行 9 で記述されている．
- 行 2 の実行時にすでに他のスイッチがオンである場合は，行 9 の条件から，「s becomes on」を真にすることはできないので「s does not anything」の方を真にする（つまりなにもしない）．これは期待どおりの行動である．なお，ここでは，TurnOn 指示の操作が特別の優先度を持たないと仮定したので，FNL の語用規則 ⑤ 現実モデルの「許容変化性」の条件により上記のように行動することになるが，TurnOn 指示が高い優先度を持つ場合は，当スイッチをオンにし，かつ，（行 9 の恒常的制約条件を順守するため）すでにオンであるスイッチをオフにする，という行動をとることも FNL 意味論上は可能である（ただし，5.2.1 項で述べるように，主体の意思の優先度については未実現である）．
- 「操作しなかったスイッチの状態は（モデル構成上必要がないかぎり）変わらない」ということは FNL の語用規則 ⑤ 現実モデルの「極小変化性」の中に含まれているので，その旨を明に記述しなくてよい．

（例 4）**迷路問題**：設定された迷路をたどって，ロボットたちが衝突を避けつつそれぞれの目標地点に向かう．

［**図 5** の FNL 記述の要点］

- 迷路問題は目標（ゴール）指向記述を象徴する問題である．FNL ではそれを主体 Robot の行動目標（行動本性；行 1〜2）と恒常的制約条件（行 3〜行 8）を記述するだけで表現している．
- 行 9 の start 指令が投入されると Robot が生成される（行 10）が，その時点でその Robot は行 1〜2 の行動本性に従って行動し始める．

```
 1: @ Robot:r ==
 2:    r becomes at r's Goal .
 3: true ==
 4:    for all Robot:r, for all Place:p;
 5:       not ( r is at many(2) Place )  and
 6:       not ( many(2) Robot are at p ) and
 7:       ( r becomes at p -> ( r is not at p ->
 8:            r is at some Place which is by p ) ) .
 9: start ( the Robot:r,the Place:s,the Place:g ) ==
10:    r becomes existent  and
11:    r becomes at s  and
12:    r's Goal becomes g  .
```

図 5　迷路問題の FNL 記述
Fig. 5　Maze problem written in FNL.

- 図 5 の迷路の問題記述は暗に計画挙動（プランニング）を含んでいる．すなわち，図 5 の行 2 では，「いずれ必ず “the Robot:r is at r's Goal” を真にする」と言明しているので，FNL 言語実行システムは，true 定義文で記述されている恒常的制約条件をみたしつつその言明を達成するべく動作させる．たとえば，行 7〜8 の制約条件から，Robot は，ある地点 p に移動するにはその前に p の隣（by）の地点に移動するという動作を行うことになる．この動作は FNL でいう現実的モデルを構成する過程であり，本問題の記述者が期待する動作とも一致する．本記述の実行時にそうした計画挙動を行うことは，FNL の意味論と語用論から説明できる．<注> なお，5.2 節で述べるように計画挙動を行うプログラム導出は未実現である．
- 本記述は，Robot の移動中に目標地点や迷路設定が動的に変更されるような開放環境においても通用する．

（例 5）**図形定義問題**[24]：「図」を上位実体として，その下位に，「四角形」，「平行四辺形」，「菱形」，「長方形」，および「正方形」という実体の種を順次定義する．

［**図 6** の FNL 記述の要点］

- FNL でもまた，実体の階層性（これは継承につながる）を表現することができる．その表現方法も，それ特有の新たな基本詞を導入することなく，FNL の通常の記法を援用して記述できる．

© 2011 Information Processing Society of Japan

```
 1:  Quad:q ==
 2:     Fig:q where
 3:       ( q's V1 + q's V2
 4:         + q's V3 + q's V4 = 0 ) .
 5:  Parallellogram:p ==
 6:     Quad:p where
 7:       ( p's V1 + p's V3 = 0 ) .
 8:  Rhombus:r ==
 9:     Parallelogram:r where
10:       ( abs(r's V1) = abs(r's V2) ) .
11:  Rectangle:e ==
12:     Palallelogram:e where
13:       ( e's V1 * e's V2 = 0 ) .
14:  Square ==
15:     Rhombus and Rectangle .
```

図 6　図形定義問題の FNL 記述
Fig. 6　Figures definition problem written in FNL.

4. 実 行 方 式

本章では，当言語の実行可能性を示すために我々が試作した FNL 言語システムの実行方式を述べる[25]．

4.1 FNL の実行方式

FNL 言語システムの構成を**図 7** に示す．FNL 言語システムは，おおきく，翻訳系，論理合成系，目的言語展開系，および実行系の 4 つのコンポネントからなる．言語 FNL で記述した問題仕様をこのシステムに投入すると，最終的には，目的言語（現実装では C 言語）で書かれたプログラムがデータ宣言部も含めて自動的に導出される．この後それをコンパイルすれば，ただちに実行可能になる．以下これらの各コンポネントについて要点を述べる．

翻訳系：言語 FNL の糖衣構文で記述された問題仕様を正規の内部表現に変換する．これにより，後続の論理合成系以降，問題仕様を形式的に処理することが可能になる．

論理合成系：内部表現された問題仕様を項書き換え規則を再帰的に適用することにより，通常のプログラム言語とオペレーティングシステム（以下 OS と略記する）上で実現可能な中間構文子と中間命令だけからなる中間プログラムに変換していく．1 つの書き換え規則は言語 FNL の 1 つの基本詞を消去し中間構文子に変換するものであり，「not 消去」，「and 消去」，「or 消去」，「for all 消去」，「for some 消去」，「for many 消去」，「=> 消去」，「when 消去」，「while 消去」，「# 消去」などの書き換え規則がある．書き換え後の中間プログラムで使用する中間構文子としては，「;（逐次実行）」「if-else（条件判定）」，「repeat（無限反復）」，「forall（実体群の全走査）」，「break（反復からの脱出）」，「return（復帰）」などがあり，中間命令としては，「startproc（プロセス起動）」，「endproc（プロセス停止）」，「create（実体生成）」，「delete（実体消去）」，「setrel（関係設定）」，「check（関係判定）」，「delrel（関係消去）」，「setattr（属性設定）」，「action（行為）」「lock（排他的実行権獲得）」，「unlock（排他的実行権解放）」，「post（世界状態変化の通知）」，「wait（世界状態変化の待機）」，および「noop（無動作）」などがある．主要な書き換え規則を**図 8** に示す．また付録 A4 に論理合成例を付記した．

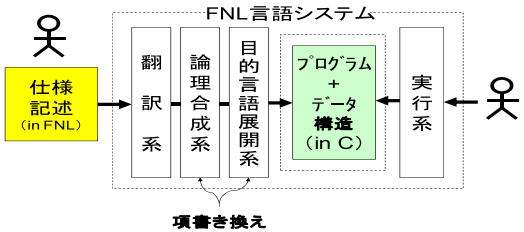

図 7　FNL 言語システム
Fig. 7　FNL language system.

目的言語展開系：上述のように，論理合成の結果出力される中間プログラムは，通常のプログラム言語と OS 上で実現可能な中間構文子と中間命令だけからなっている．これを目的プログラムに展開する．そのとき，その目的プログラムでは実体，実体の属性，および実体間の関係をどのようなデータ構造で表すかを統一的に決めておけば，目的プログラムの手続き部と同時にそれが必要とするデータ宣言部も自動的に生成できる．たとえば現実現方式では，各実体および各関係は C 言語の構造体として表す．そして，実体の属性はその実体の構造体のフィールドとし，関係については，その関係が n 項関係であれば，その関係の構造体には n 個の項フィールドを設けるようにデータ構造を定義している．現行の FNL 言語システムの統一的データ構造の形態を**図 9** に示す．

実行系：実行を指示された問題仕様に対応する目的プログラム（これは事前に自動生成されている）を計算機上にロードし，その動作環境を整える．その後，操作コマンドが投入されたとき（実行系は操作者に対してつねに操作コマンド読み込み状態にある），あるいは，なんらかの実体が生成されたときには，そのつど新たにプロセスを生成し，そのプロセスのもとで，その操作コマンド，あるいは生成された実体の行動本性に対応するプログラム部分を起動する．操作コマンドの場合は同時に入力パラメータを引き渡す．実行系の中には，先に述べた中間命令群の実行体も含まれている．

© 2011 Information Processing Society of Japan

```
"not(P)"
if ( "P" )
   return false
else
   return true;
```

```
"for all (A:x) P Q"
forall(A:x) {
   if ( "P" )
      if ( "Q" )
         noop ;
      else
         return false ;
   else
      noop ;
}
return true ;
```

```
"while P Q"
repeat {
   lock ;
   if ( "P" )
      if ( "Q" ) {
         unlock ;
         post ;
      }
      else {
         unlock ;
         wait ;
      }
   else {
      unlock ;
      post ;
      break ;
   }
}
return true ;
```

<注1>図中の太線枠部は，書換え対象の内部正規表現であり，それに続く細線枠部は，その書換え結果を示す

<注2> "X"(引用符囲み部分)は，Xがさらに論理合成可能な(書き替え規則適用可能な)内部正規表現であることを示す．

```
"and P Q"
if ( "P" )
   if ( "Q" )
      return true;
   else
      return false;
else
   return false;
```

```
"=> P Q"
lock ;
"P" ;
unlock ;
post ;
lock ;
"Q" ;
unlock ;
post ;
return true;
```

```
"relative(A:x,B:y)"
if (check(2,relative,A,x,B,y))
   return true;
else
   return false;
```

```
"becomes(relative(A:x,B:y))"
if (setrel(2,relative,A,x,B,y))
   return true;
else
   return false;
```

```
"becomes(existent(A:x))"
if (create(A,x))
   if (startproc(A,x))
      return true;
   else
      return false;
else
   return false;
```

```
"does(verb(A:x,B:y))"
if (action(2,verb,A,x,B,y))
   return true;
else
   return false;
```

```
"if P Q"
if ( "P" )
   if ( "Q" )
      return true;
   else
      return false;
else
   return true;
```

```
"when P Q"
repeat {
   lock ;
   if ( "P" )
      if ( "Q" ) {
         unlock ;
         post ;
         break ;
      }
      else {
         unlock ;
         wait ;
      }
   else {
      unlock ;
      wait ;
   }
}
return true ;
```

```
"# (A:x) P"
count := 0;
forall(A:x) {
   if ( "P" )
      count := count + 1 ;
   else
      noop;
};
return count;
```

<注3> それぞれの合成(書換え)結果が(それを含む)プログラム内にインライン展開すべき場合は，「return true」部分(「return false」部分)は「truth := true」(「truth := false」になる．ここで「truth」は真偽値を後続処理に受け渡すための内部変数である．それらの「return true」部分(「return false」部分)がforall節にある場合は，さらに，反復脱出のための「break」命令も付加される．

図 8　論理合成系の主要な書き換え規則
Fig. 8　Principal rewriting rules used by FNL synthesizer.

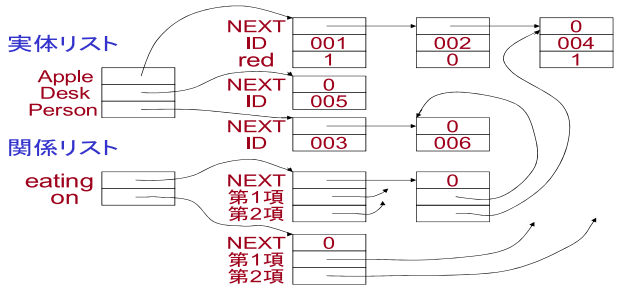

図 9　FNL 言語システムのデータ構造の例
Fig. 9　An example of data structures supposed by FNL.

4.2　FNL 実行方式の正当性

上述の FNL の実行方式（特に論理合成）が FNL 意味論に照らして正当であることについての厳密な議論は省略するが，図 8 および付録 A4 の論理合成規則および論理合成例から了解されるだろう．なお，現実的モデルの構成（すなわち問題あるいは仕様の実行）においては極小変化性を保証するなどの非単調推論を行う必要があるが，FNL では問題記述を通常の命令型プログラムに変換して実行するので，そうした推論（特に極小変化性の保証）はむしろ自然に行いやすい．

5.　評価と議論

5.1　FNL の記述性

FNL の設計方針において記述性に関していくつか要請を課したが，その充足性について以下に評価する．また，それらの要請は，具体的には，記述の簡潔さ（記述量），読解性，および仕様変更への対処の容易性などの効果をもたらすべきものであるが，それらについても実験結果を示す．

5.1.1　自然語指向

2.3 節言語仕様および 3 章記述例で示したように FNL は自然語と類似した品詞体系を持ち，構文論および意味論のいずれの観点からも，ソフトウェア記述言語の中では自然語に近いものになった．ただし，FNL の構文は自然語そのものではない．これは，FNL が技術用言語として自然語の中の論理的部分だけを反映するように意図的に自然語と形式論理言語の中間的位置を選んだ結果である．あまりに自然語に近いと技術用言語としてはかえって読み書きが難しくなることがある．

© 2011 Information Processing Society of Japan

(1)　他の自然語指向言語との比較

自然語を指向した形式仕様言語としては，TELL[9] および Controlled Natural Language (ACE)[26),27)] などがある．これらと FNL の違いを対比的にいえば，TELL および ACE は構文論的に自然語指向であり，FNL は存在論的・意味論的に自然語指向であるといえる．構文論的観点では，TELL および ACE の構文規則は，自然語（英語）そのものをベースにし，それに制限を加えたものになっている．一方 FNL の構文規則は，述語論理をベースにし，それに自然語らしい構文糖衣を施したものである．したがって構文論的には，TELL および ACE の記述は FNL に比してはるかに自然語に近く豊かな表現が可能である．一方，存在論的・意味論的観点では，FNL では，自然語の背後にある存在論を明示的に構成したうえで，それを反映した意味論を構築している．その結果，問題世界の静的構造，動的振舞い，および並行性を自然かつ統合的に記述できるものになった．その点，TELL や ACE の自然語指向は存在論にまでは及んでいない．また，ACE は現時点では動的振舞いや並行性の記述力は十分でないように見える[26)]．

(2)　自然語の曖昧性の問題

自然語指向の仕様言語の場合，通常自然語の曖昧性の問題がつきまとう．たとえば，for all などの限量詞の順序や有効範囲，all，each，および名詞の複数形の間の微妙な意味の違い，ならびに文脈に依存した意味解釈の違い等々多種の曖昧性の問題が指摘されている[28)]．しかし，ここまで述べたように FNL は，自然語そのものではなく，形式的な構文規則と意味定義を備えた形式言語であるので，そうした自然語の曖昧性の問題はまったく生じない．たとえば先の限量詞の順序については述語論理と同じく厳格に意味定義されている．また，all，any，および every は，FNL ではまったく同じ意味とされる．これはもちろん，FNL が自然語の曖昧性の問題を積極的に解決したということではなく，FNL では，なまの自然語で表現された原問題を曖昧性のない FNL 記述に翻訳するのは記述者の責任として残すという選択をしたということである．

5.1.2　推　論　性

FNL の推論性を示す代表的な例として，(a) 恒常的制約条件からの推論，(b) FNL 意味論に基づく各種推論，および (c) 型推論があげられる．このうち，(a) と (c) については 3 章記述例で見たので，ここでは (b) について，2 つ例を見てみる．

〔例 1：同時実行制御についての推論〕

先述の食事する哲学者問題（図 3）の場合，FNL 意味論に基づけば，問題記述から次のような推論ができる．

- もし行 6 の行為を複数の哲学者（Philo）p と q が同時に実行するとする．すると，その単位時区間で，

  (p does eat) and (q does eat)

  が真になることになる．
- これと行 12 の制約条件から，同じ単位時区間では，さらに

  (p does use many(2) Fork) and (q does use many(2) Fork)　　$\cdots$(A)

  も成り立たなければならない．
- 一方，行 11 の制約条件から，つねに，

  for any Fork:f {not(many(2) Philo does use f)}　　$\cdots$(B)

  が成り立っていなければならない．
- ところが，上記 (A) と (B) は両立不可能な（矛盾する）場合がある（たとえば，行 13 の制約によって，p と q が使えるフォークが合わせて 3 本しかない場合など）．
- ということは，結局，行 6 の行為を複数の哲学者（主体）p と q に同時に実行させることは無条件には許すことができない，ということになる．

このようにして，問題記述だけから FNL 意味論に基づいて推論することにより，行 6 の行為は，なんらかの同時実行制御（複数主体が同時に行為することから生じる矛盾を避けるための制御；それは排他制御のような悲観的制御に限る必要はない）が必要であることが分かる．したがって FNL では，問題記述者にそれについて明示的に記述させることはしない．これが FNL の推論性の一例（同時実行制御の推論）である．

なお，このように，FNL 意味論では，

- 単位時区間行為の概念がデータベースのトランザクション理論におけるトランザクションの概念の役割を果たし（逆に，どんなトランザクションも FNL における単位時区間行為として表現できる），
- 単位時区間行為の同時実行可能性はその同時実行無矛盾性として定義する．

この同時実行可能性の定義は，トランザクション理論における直列可能性（serializability）の基準を真に含むものになっている．つまり，トランザクション理論においては，2 つのトランザクション P と Q の同時実行可能性を，P と Q の直列可能性によって定義したが，FNL では，2 つの単位時区間行為 P および Q の同時実行可能性は，「{P and Q} の無矛盾性」として直接的に定義する．この定義に従うと，たとえば，{x := y and y := x} という同時行為（x と y の値の同時交換）は，直列可能性の基準では許されないが FNL では許される，という違いが生じるが，並行（並列）環境を前提にすると，FNL の同時実行可能性

© 2011 Information Processing Society of Japan

```
 1:  withdraw ( the Account:a, the Number:n ) ==
 2:     a's balance := a's balance - n .
 3:  deposit ( the Account:a, the Number:n ) ==
 4:     a's balance := a's balance + n .
 5:  transfer ( the Account:a, the Account:b,
 6:              the Number:n ) ==
 7:     withdraw ( a, n )  and
 8:     deposit  ( b, n ) .
 9:  true ==
10:     for any Account:a ( a's balance >= 0 ) .
```

図 10　銀行口座問題の FNL 記述
Fig. 10　Bank accounts problem written in FNL.

の定義の方が自然であると思われる．

次に，上述のような同時実行可能性についての推論を FNL 言語システムがどのように自動的に行い，そしてその推論に基づいて FNL 言語システムがどのようなプログラムを自動生成し，どのように実行制御するかという疑念が生じるだろう．

これについては，現行の FNL 言語システムでは以下の素朴な方式をとっている．

- 単位時区間行為はすべて，同時実行矛盾の可能性があると悲観的に判断する．
- そして，プログラム生成時には，それらの単位時区間行為に対応するプログラム部分の前後にはかならず所定の lock 命令および unlock 命令を挿入する．
  （このように，FNL 言語システムが lock 命令および unlock 命令を自動的に挿入するので，それにともなうデッドロック回避の考慮（たとえば，すべてのロックに全順序を設定したうえで 2 相ロック方式を採用するなど）も，個々の開発者に依存することなく，FNL 言語システムの責任のもとで漏れなく実施することができる．ただし現行では，ロックは FNL 言語システムに 1 個だけにしているので，こうしたデッドロック回避の考慮はしないでもよい．）

この方式により，（過剰防備ではあるが）安全な（正しい）プログラムは自動生成できる．そして，さらに実行性能の向上を図る場合にも，FNL 意味論では（先述のように）同時実行制御というものを同時実行矛盾の回避という最も一般化した位置づけでとらえているので，データベースのトランザクション分野の研究成果を援用するなど，より良い制御方式に柔軟に拡張できる枠組みになっている．

〔例 2：待機制御についての推論〕

端的な例として，FNL で記述した銀行口座問題[29]（**図 10**）を考える．この場合も，先の例と同様にして，withdraw 行為と deposit 行為（あるいはそれら自身どうし）について同時実行制御が必要であることは，行 2 と行 4 について FNL 意味論に基づいて推論することにより判明することであるので（行 2 と行 4 を同時に実行すると，同じ口座の残高（balance）が，異なる 2 つの値になるという矛盾が生じうるから），それについて明示的な記述は不要である．そしてさらに，制約条件（行 10）に矛盾してはならないという要請から，残高が十分にないときは withdraw 行為や transfer 行為は待機するという制御も自動的に行われる．これもまた FNL の推論性の例（待機制御の推論）である．

```
@ Philo:p ==
  ...
  while ( p is hungry ) {
    { ( p does eat ) and
      ( p does use the Fork:f1 ) and
      ( p does use the Fork:f2 ) }
  }
  ...
```

図 11　食事する哲学者問題のデッドロックフリー性（その 1）
Fig. 11　Dining Philosophers Problem's deadlock-freeness (1).

〔補足 1：食事する哲学者問題のデッドロックフリー性〕

食事する哲学者問題の FNL 記述（図 3）の実行時にフォークについてのデッドロックを生じないことについて補足する．

いまの場合問題となる図 3 の行 5–6 だけに注目する．行 12 の恒常的制約条件を加味すると（実際には行 11 および行 13 も関係するが簡潔のためここではそれらは無視する），ここはまず，**図 11** のように論理合成される（5.2 節で述べるように恒常的制約条件のプログラム組み込み方式は未検討なので実際の合成結果がこのとおりになるか否かは未確定であるが，FNL 意味論から，does eat と 2 つの does use the Fork が and で結合されることには間違いはない）．さらに，これに，while 消去規則，and 消去規則，および does 消去規則を適用すると，結局**図 12** のように書き換えられる．

このようにして，最終的には，Philo たちは 2 本のフォークを逐次的に使用するようなプログラムが生成される．しかし，それらの命令が lock 命令と unlock 命令で囲まれているので，たとえ 2 人（それを P1 と P2 とする）が同じ 2 本のフォークを使用し，かつ，一方はフォーク f1 から使用し，片方がフォーク f2 から使用するように実行するとしても，それらが相互割込みして，$\mathrm{action}(2, \mathrm{use}, \mathrm{Philo}, \mathrm{P1}, \mathrm{Fork}, \mathrm{f1}) \rightarrow \mathrm{action}(2, \mathrm{use}, \mathrm{Philo}, \mathrm{P2}, \mathrm{Fork}, \mathrm{f2}) \rightarrow \cdots$ のような順序で 2 人のプログラムの命令が実行されることはありえず，フォークについてのデッドロックが生じることはない．なお action 命令は，世界状態になんら痕跡を残さな

© 2011 Information Processing Society of Japan

```
@ Philo:p ==
  ...
  repeat {
     lock;
     if (check(1,hungry,Philo,p)) {
        if (action(1,eat,Philo,p)
           if (action(2,use,Phlo,p,Fork,f1))
              if (action(2,use,Philo,p,Fork,f2))
                 truth := true;
              else
                 truth := false;
           else
              truth := false;
        else
           truth := false;
        if (truth = true) {
           unlock;
           post;
        }
        else {
           unlock;
           wait;
        }
     }
     else {
        unlock;
        post;
        break;
     }
  }
  ...
```

図 12　食事する哲学者問題のデッドロックフリー性（その 2）
Fig. 12　Dining Philosophers Problem's deadlock-freeness (2).

い（何の構造体も残さない）命令であるので，action 命令実行後は，not(p does use) に見える．

FNL ではこのように，lock 命令と unlock 命令による排他実行によって and の同時実行性（より広くは単位時区間行為の原子性）を実現している．つまり，ある主体がこれらの lock 命令と unlock 命令の間で複数の行為を逐次的に実行しても，他主体からは，それらの行為の間隙で世界状態を参照&更新することはできないので，あたかもそれらの複数行為が同時実行されているのと同じに見えることになる．

〔補足 2：単位時区間行為とトランザクション〕

FNL 意味論における単位時区間行為とデータベース分野におけるトランザクション概念を同一視できることについて補足する．

まず，単位時区間行為はトランザクションと見なすことができる．というのは，FNL 意味論における単位時区間行為とは，世界の事前状態の参照とそれに基づく事後状態への更新をいわば一瞬のうちに（それ以上分割されえない単位時区間で）行うものである．また，その単位時区間行為は実行するかしないか（真か偽か）のどちらかしかなく，実行しないということは無動作と同じになる．以上はトランザクションの原子性（Atomicity）に対応する性質である．次に，（ある行為主体が）単位時区間行為を実行するということはその行為文が意味すること（すなわち世界の事前状態から事後状態への更新）を真にする（正しく実行する）ことであり，これは多主体並行動作環境下のどの主体の単位時区間行為についても適用される意味規則である．どの主体についてもこの意味規則が守られるということ，すなわち，並行動作しているどの主体についてもそれが意図した行為が正しく実行される（他主体の行為によってそれが阻害されることはない）ということはトランザクションの隔離性（Isolation）に対応する性質である．そして，FNL 意味論ではシステムを実行するということはその仕様のモデルを構成することであるので，実行された単位時区間行為の終端時点でも，世界の状態に矛盾が生じているということは許されない．これは，トランザクションの整合性（Consistency）に対応する性質である．なお，トランザクションの永続性（Duarability）は，論理的領域外の事項であり，FNL 意味論では関知しない．

逆に，トランザクションは単位時区間行為と見なすことができる．なぜなら，トランザクションとは，一般に，あるプロセスがある事前の世界状態に基づき適切な事後状態に更新する処理のことであり，そのとき元来他のプロセスの介入を期待していない（むしろ排除する）．したがって，本来トランザクションの意図は世界の事前状態から事後状態への（一瞬の）更新として記述可能であり，FNL ではまさにそれを単位時区間行為として記述する（現行のプログラムでは，プログラム言語の制約から，それを複数の実行ステップとして記述するのが通常であるが）．つまり，どんなトランザクションも FNL の単位時区間行為として記述することが可能である．

以上から，FNL の単位時区間行為とトランザクションとは同一視できることが分かる．そしてどの行為が単位時区間行為であるかは FNL では自然に識別可能であるので（たとえば，FNL 意味論から「becomes(P)」，「becomes(P) and becomes(Q)」，および「for all A:x becomes(P)」などは単位時区間行為であるが，「becomes(P) => becomes(Q)」や「while P becomes(Q)」などはそうではないことが分かる），通常のプログラム言語のように特別の構文子を用意してトランザクションの区切りを指定させる必要がない．

なお，ここでの論述のように，これまでトランザクションについての議論自体が「ACID」のような非形式的に記述された基準を援用して議論せざるをえなかったが，FNL ではそれ

© 2011 Information Processing Society of Japan

表 2　存在論の対照比較
Table 2　Comparison of ontologies.

FNL	KAOS	Tropos
世界（問題）	Goal	Goal
実体	Entity	Resource
関係	Relationship	Belief
事態	Event	
行為主体	Agent	Actor Dependency Capability
行為（脈絡）	Action Scenario	Plan
論理（恒常的制約）	Constraints	

を単位時区間行為という簡潔かつ形式的に定義された概念を用いて議論することが可能になる．

5.1.3　存　在　論

要求仕様言語 KAOS [11] および Tropos [30] は存在論を明示している数少ない言語である．ここでは FNL とそれらの存在論の対照比較を**表 2** に示す．

表 2 のように，FNL の存在論は最上位のレベルでは他 2 者のそれとおおむね対応関係がある．相違点があるのは，FNL がまずは最小普遍の存在論（最上位オントロジ）に立脚しようとしているのに対し，他 2 者はさらに特定の意図（KAOS の場合はゴール指向，Tropos の場合はエージェント指向）を強調していることによると考えられる．

5.1.4　統　合　性

FNL では，世界の静的構造，動的振舞い，および並行性を統合した存在論を明示し，それを言語仕様に反映したという意味で統合的である（そもそも「静的構造」，「動的振舞い」，および「並行性」という自然語概念が FNL の意味論の中では有機的に統合され形式化される）．それによりこれら 3 要素を含む問題（実用システムの多くはそれらが混然とした問題である）が一様に記述できる．なお，こうした統合化を図るものとしては CSP-OZ [31]（CSP と Object-Z の連携）や csp2B [32]（CSP と B の連携）などがあるが，それらが既存の言語を複数連携させることによりそれを行うのに対し，FNL は 1 つの言語として統合的である．前者は現実的なアプローチであるが，FNL の統合化はより有機的であるといえる．

5.1.5　素朴&直接性

FNL の言語要素の中には，問題世界の対象を直接指示するものだけがあり，2.1 節にあげたようなメタ的言語要素は厳しく排除されていることは明らかだろう．これは現場技術者にとっての記述道具として，記述&読解性および習得性を高める特徴であるといえる．

表 3　記述量の比較
Table 3　Comparison of the volume of descriptions.

記述例題	記述行数			FNLの記述量軽減の要因と効果			
				推論性			直接性
	FNL	他言語		恒常的制約	FNLの意味論	型の推論	メタ用語
住所録	5	16	Z [20]	0	10%	50%	10%
		11	VDM [20]	0	15%	25%	15%
食事する哲学者	13	29 + α	CSP [22]	50%		0	0
スイッチ	8	36	CafeOBJ [23]	20%	10%	35%	0
座席予約	12 [付録A5]	36	B [33]	2%	2%	20%	40%
Package Router	64 [付録A6]	139	Z [34]	0	15%	25%	0

5.1.6　問題純粋性（実現隠蔽性）

これについては，原問題と問題記述の対応関係を分析することによって評価可能である．FNL 記述の以下の特徴は，その問題純粋性を示すものである．

(a)　原問題にある制約条件がほぼそのまま FNL 記述の true 定義文（恒常的制約条件）として記述される．

(b)　原問題に現れない概念（特に実現概念）は FNL 記述上にも現れない．

上記 (a) については，3 章の「食事する哲学者問題」や「スイッチ問題」の true 定義文記述がその例である．(b) については，3 章のいずれの例題もその例になっている．

なお，「それでも原問題と FNL 記述の間には一定の構文的あるいは意味的距離がある」という議論はありうるが，その距離は，算数の文章題とその代数的定式化の間の距離に比しても小さいものであり，受容可能であると考えている．

5.1.7　簡　潔　性

表 3 は，いくつかの例題を FNL と他の形式仕様言語で記述した結果を集計，比較したものである．ここに示すように，FNL は他言語に比して記述量が少なくなっている．それには，推論性と直接性が最も効果を発揮している（なお，プログラム言語と比較すると記述量の違いはさらに大きくなる）．

© 2011 Information Processing Society of Japan

表 4　読解性の実験結果
Table 4　The result of an experiment of readability.

	参考にした記述	正解者	不正解者
OCL記述 （7行）	0	0	0
Alloy記述 （2行）	6	3	3
FNL記述 （6行）	37	23	14

5.1.8 読　解　性

これには，FNL の自然語指向および素朴&直接性が関係してくる．

［読解性評価その 1］同じ制約条件を OCL，Alloy[35]，および FNL の 3 つの言語で（つまり 3 通りに）記述したもの（付録 A2 参照）を被験者に示し，それらの詳細な文法知識を与えないまま，最も読みやすいものを任意に選び，その意味内容を読み取らせる実験を行った．実験結果を**表 4** に示す．大多数の被験者が．読解のために FNL 記述を選択し，かつ一定の正解率を示している．この実験結果は，FNL の自然語指向が一般技術者にとって読解性が高いことを示唆している．

［読解性評価その 2］一般技術者が既存の形式仕様言語に感じる違和や困難のいくつか（下記の例）が FNL では解消している．これは FNL が自然語に基づく記述文体を指向したことの効果であると考えられる．

（例 1）事態の変化（「〜になる」）を集合の変化として表現する違和
（例 2）能動的実体と受動的実体の区別がない違和
（例 3）参照と行為の区別がない違和
（例 4）変わらないことを明示的に記述する違和[18]
（例 5）単純な事態の記述に巾集合や関数型が現れることの違和
（例 6）語彙定義が主要部に見える仕様記述の違和
（例 7）時点論理における変化構造の記述の困難

なお，言語習得の容易性は読解性と高い相関があると予想されるが，これについても，今後別の実験でそれを実証したいと考えている．

5.1.9 変更容易性

先に簡潔性の評価で見たように記述量が少ないこと自体が変更容易性に有利に働くと考えられるが，さらに記述内容の観点から FNL の変更容易性について見てみる．

〔例：銀行口座問題（図 10）の変更〕

銀行口座の原問題が，以下のように拡張された場合[29] を考える．

拡張：2 つの口座を指定し残高が十分なほうの口座から引き出すこと（選択引き出し；withdraw2）ができるようにする

このとき FNL では，図 10 の記述に対して新たに

```
11:  withdraw2 ( the Account:a, the Account:b, the Number:n ) ==
12:      withdraw ( the Account:a, the Number:n )  or
13:      withdraw ( the Account:b, the Number:n ) .
```

を追記するだけで対応可能である．そのとき，同時実行制御および待機制御も原問題を自然に拡張した形で行われる．つまり，withdraw2 と deposit や transfer の間の同時実行制御，ならびに，どちらの口座にも十分な残高がない場合 withdraw2 全体を待機させる制御などが自動的に行われる．一般に，変更容易性に関しては，原問題の要件が 1 つ変わったときには問題記述上も 1 カ所変更すればよいのが理想的であるが，FNL の場合は特にその問題純粋性と推論性（これにより論理的に重複する記述が避けられる）から，それに近くなっている．

5.2　FNL の実行可能性

5.2.1　FNL の現時点での実現実績と制限

表 5 に，上述の実行方式の現時点での実現実績を示す．

表 5 に示したように，① 叙述詞（is，becomes，does），② 主体詞（@，プロセス起動），③ for all，for some，および for many などの限定詞，④ 時相詞 when（待機），⑤ 普通名詞，属性詞，関係詞からのデータ構造の自動生成（データベースの関係テーブル自動生成も含む）などは，4.1 節で述べた比較的シンプルな方式によってすでに実現済みである．これらは FNL の言語仕様の骨格部分であり，かつ通常のプログラム言語や述語論理を超える部分である．これらを実現したことにより，（疑念が生じがちな）当言語の全体的実現可能性は示しえたと考えている．

一方，⑥ 恒常的制約条件のプログラム組み込み，⑦ 同時実行の意味の and および for all，⑧ 選択実行の意味の or，⑨ 計画挙動（プランニング），⑩ 実体階層（継承機構），および ⑪ 行為主体の意思の優先度などについてはいまだ実現の実績はない．それらを行うのに十分な情報は FNL の問題記述の中にあり，その実現に本質的な困難はないと思われるが，極力シンプルな実行方式を検討中である．

現時点まで実現されている言語仕様に基づくプログラム自動生成の実験実績を付録 A3 に

© 2011 Information Processing Society of Japan

表 5　FNL の現時点での実現実績
Table 5　Current executability of FNL.

品詞	基本詞	現時点での実行可能性
定義詞	==	○
	true == （恒常的制約条件）	×
	実体階層の定義	×
叙述詞	is	○
	becomes	△ （「becomes(P)」における「P」が 単純事態文の場合のみ実現済； 複合事態文の場合は未実現）
	does	○
	=	○
	:=	○
普通名詞	Entity	○
	Number	○
	String	○
	Set（集合）	×
	（任意）	○（データ構造の 自動生成も実現済）
実在詞	existent	○
	nil	○
属性詞	（任意）	○（データ構造の 自動生成も実現済）
関係詞	（任意）	○（データ構造の 自動生成も実現済）
事態詞	true	○
	false	○
	（任意）	○
行為詞	（任意）	○
論理詞	not	○
	and	△ （同時実行の意味の and は未実現）
	or	△ （選択実行の意味の or は未実現）
	if　->	○

品詞	基本詞	現時点での実行可能性
限定詞	all	△ （all の同時実行は未実現； all の反復実行は実現済）
	some	○
	many	△ （many の同時実行は未実現； many の反復実行は実現済）
	the	○
主体詞	@	○
時相詞	=>	○
	while	○
	when	○
修飾詞	where	○
	who	×（whereで代用）
	which	×（whereで代用）
	whose	×（whereで代用）
助詞	:	○
	's	○
	for	○
算術詞	+	○
	−	○
	*	○
	#（集合の要素数）	○
	sum（総和）	○
固有名詞	定項	○
指示詞	変項	○

○：実現済，△：一部未実現，×：未実現
（任意）：問題ごとに記述者が導入するもの
単純事態文：論理詞and, or, if（->）および限定詞all, some, manyを含まない事態文．複合事態文とは，それらを含む事態文．

示した．なお，この実験での FNL 記述はいずれも，現段階での実現上の制限から，恒常的制約条件や同時実行の and は使用しない形で記述したものである．

〔補足：プログラムの完全自動導出について〕

FNL では，プログラム導出は完全に自動的に行う（その間アルゴリズムの指定などの人間の介入はいっさい要しない）というのが設計方針である．上記のこれまで実現実績がある部分についてもこの意味で完全自動でプログラムが導出されている．一方，上記の未実現の部分についてはもちろんその実績はないが，その場合も完全自動で行うという方針は変わらない．

```
 1: Sort ==
 2:   for all Entity:e, for all Entity:f
 3:     { ( e's Value >= f's Value ) ->
 4:       ( e becomes following f ) } .
 5: the Entity:e is following the Entity:f ==
 6:   ( e is next f ) or
 7:   ( e is next some Entity which is following f ) .
 8: true ==
 9:   for all Entity:e
10:     not ( many(2) Entity are next e ) .
```

図 13　ソートの FNL 記述
Fig. 13　Sort written in FNL.

5.2.2　FNL の実行性能

FNL が自動生成する C 言語プログラムの実行性能については，今回特別の考慮はしなかった．これは，FNL のような問題純粋な記述から，まずは実行可能かつ正しいプログラムを自動生成することを最優先の課題としたためである．ただし現行でも，FNL によって生成されたプログラムはインタプリタ方式でなくコンパイラ方式で動作すること，ならびに，for all や for some などの限定詞の処理を（限定条件をみたす対象の集合を事前に形成するというような重い処理ではなく）通常の反復処理と条件判定によって実現していることなどから（4 章図 8 の論理合成規則参照），そうでない場合ほど極端に実行性能が悪いわけではない．我々の実験では，より実用的なデータベースアプリケーションの場合には（注：FNL 記述はそのままで関係データベースアプリケーションとして実行させることもできる），むしろ，C 言語プログラム自体の実行性能より，外部のデータベースチューニングの方が問題になった．しかし，いずれにせよ，問題記述から直接的に導出した現行のアルゴリズムは今後高速化の余地はおおいにある．それは，たとえば実体や関係の検索時のハッシュ法の適用，計算済み結果の再利用，および排他区間の最適化などのような比較的初等的なことから，データベースチューニングの自動化，さらには制約問題解決アルゴリズムの高速化のような本質的難題に至るまでの広範囲な最適化が必要になる．これらについては，それぞれ関連する研究分野の成果を盛り込む形で漸進的に改善していくことになると考えている．また，より一般には，並列化による性能向上も期待したい．

〔補足：ソートのアルゴリズム〕

FNL の問題記述においてもソート（整列）に類する行為が現れることはありうる．その場合，**図 13** のように記述するのが FNL らしい書き方である．

この記述を実行するプログラムを自動導出することは，高速化の課題以前に，恒常的制約

© 2011 Information Processing Society of Japan

条件のプログラム組み込み，ならびに，for all および and の同時実行の実現と関連する課題であり，現時点では未実現である．なお，FNL の考え方によれば，たとえば表示時のソートであれば，それは問題記述（Model-View-Controller パターンでいう Model）として書くのではなく View の中で処理すべきであり，また，図 13 の場合でも，Value 属性の大小関係と Entity 間の next および following 関係がいつも一致する問題の場合は，Sort 行為を導入すること自体が不要である（next および following 関係は Value 属性によって派生定義すればよい）ので，FNL 記述の中にソートを書く機会はそれほど多くないかもしれない．

5.3　FNL の検証性

一般に，形式仕様言語には，システム検証の役割が期待されることが多い[36),37)]．しかし，FNL はすでに設計済みのシステムを後から検証するためより，むしろ，ソフトウェアシステムを新規に作成し同時にその正しさを保証したい場合に用いることを主として想定した言語であることから，これまでの形式仕様言語とは，検証に関する考え方が異なる点が多い．ここでは FNL とシステム検証の関係について議論する．

5.3.1　FNL における検証の考えかた

FNL における検証では，記述&検証&実行の一体性と内包検証指向が特徴的な考えかたである．

(1)　記述と検証と実行の一体性

FNL においては，問題記述とは，対象の問題（以降システムとよぶ）を記述しきることである．その記述が論理的に含意すること（それは FNL の意味論により正確に定まる）はすべてそのシステムの仕様であり，その逆もいえる．したがって，通常はシステム検証時に設定する検証条件（安全性条件など）もここでいうシステム仕様の一部であることになる．また，システムを実行するとは，その記述（すなわちシステム仕様）のモデルを構成すること（その記述に従って実行すること）であり，これは FNL 言語システムが，プログラム生成も含めて自動的に行う．

つまり，FNL ではシステムの記述と検証と実行の 3 者はほぼ一体のものとなる．このことから対象システムの実行が検証条件を満足しているのはほぼ自明のことになり，別途システム検証を行う必要は大きく軽減される．従来の形式手法では仕様記述とシステム検証と仕様の実行化（すなわちプログラミング）の 3 者は別々に行うことが暗黙の前提になっており，そのことから，(a) 検証条件はシステム仕様が満足すべき条件全体から見ると（重要ではあるが）その一部にならざるをえず，システム「全体」の検証をするものではない，(b) 仕様検証した後もその実現過程（プログラミング過程）で誤りが混入する可能性は残る，

```
1:  reverse ==
2:     for all Element:e
3:        e's Next becomes e's Prev .
4:  the Element:e's Prev ==
5:     the Element whose Next is e .
```

図 14　リスト反転問題の FNL 記述
Fig. 14　List reverse problem written in FNL.

などの根本的な課題が生じたが，FNL ではそういうことは起こりにくい．

〔例：リスト反転問題〕ポインタで連鎖したリストを反転する問題を考える．それを従来型プログラム言語で記述した場合，正しく反転しているか否かは即座には自明でないので，検証条件として，(a) 実行中に NULL ポインタは参照されないこと，(b) 入力と出力の連鎖の要素は一致していること，(c) 入力と出力とで「指す」関係が反転していること，などを設定しての検証が必要になるだろう[38)]．一方，FNL では同じ問題を，**図 14** のように記述することになる．この記述は，原問題をそのまま記述したものに近く，これから FNL 言語システムがそれを実現するプログラムを自動生成するという前提なら，上記の検証条件 (a)(b)(c) を充足していることは，ほぼ自明である（ここで，図 14 の行 3 は，すべての Element:e について，単位時区間で同時に実行されることに注意されたい）．

(2)　内包検証指向

モデル検査法[39)] に代表される検証手法は，対象システムを状態モデルに展開し全件検査する手法である．それは，どのように記述されたシステムでも計算機パワーにより検証できるという万能性を持っているが，それと引き換えに，状態爆発の困難のような難問をかかえている．

その点，FNL は，問題（システム）記述を最も重要視する．つまり，「原問題の意図（安全性，生存性，および目標などの各条件）をそのまま直接的に，かつ良形に記述すること」により，当初から不具合を回避すること，そして極力，「その問題記述上でシステムの正当性を検証してしまうこと」を重視する．その意味で，モデル検査法やいわゆるテストにおける検証の考えかたが外延検証指向であるとすれば，FNL は問題記述からその意味を読み取り検証することを重視する内包検証指向であるといえる（なお，「原問題の意図をそのまま直接的に，かつ良形に記述すること」については，仕様記述の良否の問題として別途詳細に議論すべきテーマであるが，その議論は別稿にゆずりたい）．

〔例：飲酒する哲学者問題[40)]〕複数の哲学者が飲酒と思考を繰り返す；のどが渇いたときは複数の銘柄の酒ビンを同時に飲みたくなる；同じ酒ビンを複数人が同時に飲むことはできないし，自分の近くにある酒ビンしか飲めない；という，食事する哲学者問題をさらに一般

© 2011 Information Processing Society of Japan

```
 1:  @ Philo:p ==
 2:     while true;
 3:       { while ( p is thirsty )
 4:            for all Brand:r which p is wanting
 5:               p does drink some Bottle
 6:                           whose brand is r } and
 7:       { while ( p is not thirsty )
 8:            p does think } .
 9:  true ==
10:     for any Philo:p, for any Bottle:b;
11:       not ( many(2) Philo does drink b ) and
12:       ( p does drink b -> b is near p ) .
```

図 15　飲酒する哲学者問題の FNL 記述
Fig. 15　Drinking philosophers problem written in FNL.

化した問題を考える．この問題においてたとえばデッドロックが起きないことを検証するとき，もし本問題がなんの規制もなく自由にプログラム記述されている場合は，その状態数や経路数は容易に増大しモデル検査時にも工夫が必要になるだろう（並行動作する哲学者が 5 人，同時に欲する銘柄 3 種類，酒ビン全 20 本（10 銘柄 2 本ずつ），酒ビンはすべて 1 カ所に配置，各哲学者は 1 度ずつのどが渇く；という単純な場合でも，人の洞察をなんら加えない場合，全経路数は 10 の 17 乗のオーダになる）．

一方，FNL では同じ問題は**図 15** のように記述する．この記述は，原問題をそのまま記述したのに近く，外延検証を行うまでもなく，デッドロックの不具合が生じないことは，基礎行為「does drink」の一瞬性（単位時区間で真となり次の時区間では偽にもどる）と「for all」の同時性という FNL の意味規則のもとでは，問題記述からほぼ自明である（5.1.2 項の補足 1 と同様）．その他の安全性条件が守られていることも，恒常的制約条件（行 11〜行 12）から明らかである．

5.3.2　FNL におけるシステムの正当性の保証

FNL で記述されたシステム全体の正当性がどのように保証（検証）されるかを**表 6** に要約する．以下これについて議論する．

(1)　問題レベルの正当性と実現レベルの正当性

FNL では，システムの正当性（あるいは不具合）を，FNL 意味論を境界として，問題レベルのそれと実現レベルのそれとに明確にわけることができる（**図 16**）．

(a)　問題レベルの正当性：FNL 意味論という基盤の上で，問題（システム）記述が原問題（要求）と一致しており，かつ整合的であること．

(b)　実現レベルの正当性：FNL 言語システムが FNL 意味論を正しく実現していること．すなわち，問題記述からのプログラム自動生成と FNL 言語システムの実行系とがあいまって，FNL 意味論を正しく実現したものになっていること．

表 6　FNL におけるシステムの正当性の保証
Table 6　Guarantee of correctness.

正当性		FNLでの保証	
		問題レベル	実現レベル
一致性		○（原問題との一致性）	◎（問題記述との一致性）
整合性	安全性	◎	◎
	生存性	○	◎
	公平性	◎	◎

◎：FNL言語システムがその充足性を完全に保証する
○：FNL言語システムがその充足性を弱い意味で保証する．
　人知や静的解析ツールを援用しての強い意味での保証もしやすい

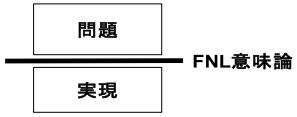

図 16　問題と実現と FNL 意味論
Fig. 16　Problem, implementation and FNL semantics.

この分類に従うと，先に述べたデッドロックも，問題レベルのそれと実現レベルのそれとにわけて考えることができる．たとえば食事する哲学者問題において，もし，哲学者が 2 本のフォークを 1 本ずつに分けて使用し始めるという問題記述であったとすればそれに関するデッドロックが生じうるが（図 3 の記述はそうでないのでそのデッドロックは生じなかった），その場合のデッドロックは上述の意味での問題レベルのデッドロックである．一方，食事する哲学者問題の実行時に lock 命令および unlock 命令を使用すること（実は，並行動作を仮定すれば，3 章の例 1〜例 4 のどの例題も実行時には lock 命令および unlock 命令を使用しての排他制御が必要である）に関して生じうるデッドロックは実現レベルのデッドロックである（そもそも lock 命令および unlock 命令は問題記述上には現れない概念なので，それが実現レベルのデッドロックであることは明らかだろう）．実現レベルのデッドロックは FNL 言語システムが自動的に回避すべき（かつそれは可能）であるが，一方，問題レベルのデッドロックについては，FNL 言語システムがそれを自動的に回避することはできない（極端な場合，問題自体がデッドロックが生じることを模擬するものであるかもしれない）．

© 2011 Information Processing Society of Japan

(2)　問題レベルの正当性の保証

(2-1)　原問題と問題記述の一致性の保証

原問題と問題記述の一致性の確認はシステム検証の出発点である．もちろんその機械的な検証は原理的に不可能であるが（この最終的検証は人知によるしかない），FNL は原問題に近い形で仕様を記述できるので，その一致性（あるいは不一致性）がより確認しやすいといえる．

さらに，原問題に近い形で記述できることは，もとから不具合のない問題記述に誘導されやすいという利点もある．たとえば FNL では and（同時実行の論理詞）を使用できるのがその一例である．食事する哲学者問題（図 3）や飲酒する哲学者問題（図 15）は，この and を使用することにより，哲学者たちは必要なフォークや酒ビンを 1 度に確保するように記述できるので，それらを 1 つずつ確保する場合に起きうる問題レベルのデッドロックを避けることができている．この場合，それらを 1 つずつ確保するように書くこともできるが，まさにデッドロックを生じさせることが目的でないかぎり，あえてそうする理由はないはずである．むしろ，1 つずつ確保するように書くことは，原問題を適切に記述していない（過剰に具体化している）ことに相当するだろう．なぜなら原問題では通常そこは「哲学者が食事するには 2 本のフォークが必要である」というだけの条件であるだろうからである．

(2-2)　問題レベルの安全性（safety property）の保証

まず，問題レベルの矛盾は起こさないという基本的安全性は FNL 言語システムが保証する．さらに，各問題に固有の安全性条件については，システム全体の安全性条件は恒常的制約条件「true == P」として記述し，各主体の行為に関する安全性条件は，各主体の行動記述の中に「while P Q」，「when P Q」，および「if P Q」の形で記述するが，これらいずれの場合も，FNL 言語システムがそれらの充足性を保証する．

上述の基本的安全性および恒常的制約条件の充足性の保証は以下のようにして行われる（「while P Q」，「when P Q」，および「if P Q」の安全性保証については自明であろうから省略する）．

① 問題レベルの矛盾は起こさないという基本的安全性：これは，FNL 言語システムの仕様実行中のどの時点における世界状態（すなわち解釈）も FNL の中では 4 章図 9 のようなデータ構造で表現することによっておのずと保証される．つまり，ここで 1 つの構造体は 1 つの基礎関係を表現しており，構造体が存在することは対応する基礎関係が真であることを表し，構造体が存在しないことは対応する基礎関係が偽であることを表している．したがって，ある基礎関係が真かつ偽であると解釈することはありえない．また，基礎属性は，構造体の要素（メンバあるいはフィールド）として表現されるが，そのデータ構造としての特性から，1 つの基礎属性が同時に異なる 2 値をとることはありえないので，基礎属性についても解釈が矛盾することはありえない．＜注＞ここで「基礎関係」とは派生定義されているのではない関係のことである．たとえば図 3 において，「by」は基礎関係であるが「near」は基礎関係ではない（派生関係である）．派生関係は世界状態を表すデータ構造として直接表現されることはない．「基礎属性」（派生属性）についても同様である．

② どの時点での世界状態（解釈）も恒常的制約条件を充足していること：これは，FNL 言語システムの以下のような動作方式によって保証される．

- なんらかの行為主体の行為を実行すると世界の状態（解釈）が変わるが，そのとき，変更後の世界状態（解釈）のもとで恒常的制約条件が成立しているか否かを判定し，成立していない場合は，世界状態を元に戻し，その行為を待機させる

〔補足 1：恒常的制約条件の充足性判定と一階述語論理の決定不能性〕

ここで述べた恒常的制約条件が成立しているか否かの判定は，「ある解釈のもとで」ある論理式（すなわち恒常的制約条件）の真偽を判定するものである．これは，（与えられた論理式が）「どんな解釈のもとでも真」であるか否かを判定すること（恒真性判定）の不可能性をいう一階述語論理の決定不能性に反することではなく，もちろん実行可能なことである．

〔補足 2：恒常的制約条件と生存性条件〕

FNL の恒常的制約条件は，どの単位時区間（いわばどの瞬間）でも成立しているべき条件を記述するものである（構文論的には，恒常的制約条件は単位時区間行為文であり，その中に主体詞（@）や時相詞（=>，while，when）を含めることはできない）．したがって，たとえば「つねにいつかは Q である」というような生存性条件を恒常的制約条件として記述することはできない．「つねにいつかは Q である（にする）」ことを保証したい場合は，「@Proc:p == while true becomes(Q)」のように，それをなんらかの主体の行動本性として記述することになる．つまり，FNL では，保証すべき生存性条件は，事後に観察点検すべき条件としてではなく，積極的に充足すべき事前の行為目標として記述する．

なお，上記の@Proc:p の記述とともに，「true == not(Q)」なる恒常的制約条件が記述されている場合は，@Proc:p の「becomes(Q)」の行為は永久に実行されず待ち状態になる．すなわち，この場合は，安全性条件と生存性条件が矛盾しているわけだが，そういう場合は，安全性条件は充足されるが，生存性条件は充足されないということになる．これは次項で述べる「FNL でも生存性については弱い意味でしか保証できない」ということの 1 つの例になっている．

© 2011 Information Processing Society of Japan

〔補足 3：CTL との関係〕

FNL の「true == P」は CTL[36] における「A□P」（「A」は「どの経路についても」の意の経路演算子）にほぼ対応する．というのは，「true == P」とは，P がつねに（したがってどの主体のどの行為においても）成り立っていることを意味するが，それは，CTL における経路概念が FNL における行為主体関数（すなわち時区間系列上での主体の順列）と同一視できることを思い起こせば，P が時区間系列上でのどんな主体の順列においても（つまりどんな経路においても）成り立っている（すなわち A□P である）ことをいっているからである．なお，FNL における「true == P」の中の「P」は事態文でなく行為文であることに注意されたい．これは FNL が基本的に，行為を単位とした時区間論理であることからくる．したがって時点論理である CTL の「A□P」（ここの「P」は事態文である）とまったく同一であるわけではない．また，主体の行動本性内に書かれた「=> becomes P」あるいは「=> does P」は，（後述の公平性を仮定したとき）CTL における（その主体の行動が開始された時点での）「A ◊ P」に対応する．

(2-3)　問題レベルの生存性（liveness property）の保証

主体の行動本性記述の中の「=> becomes P」あるいは「=> does P」は，P がいずれ成立するという生存性条件を直接的に記述したものである（なお，FNL の生存性条件の極端な場合として目標条件とよぶべきものがある．3 章記述例の図 5 迷路問題における「the Robot:r becomes at r's Goal」がその例である）．

ある主体記述の中に「=> becomes P」あるいは「=> does P」と記述しておけば，それらを真にすることが恒常的制約条件などに照らして矛盾をきたさないかぎり，いずれ「becomes P」あるいは「does P」が成り立つこと（いわば，問題レベルの「弱」生存性）は，FNL 言語システムが保証する．

しかし，それらを真にすることが永久に矛盾をきたす場合（<注> デッドロックはその典型例であるが，さらに単純なケースもある；先述 (2-2) の補足 2 参照）は，FNL 言語システムはそれらの生存性を保証することができない[*1]．そういう状態になることがありうるか否かについては，問題記述を事前に静的解析するかモデル検査をする必要がある．ただし，静的解析をするときも，FNL の場合は，以下に示すような性質から，その有効な静的解析がしやすいといえる．

- 静的解析に必要な情報がすべて問題記述上に（FNL という一様な記述形態で）存在する．かつその問題記述は一階述語論理記述にも翻訳可能であるので，適切な定理証明器と連携させて解析することも可能であるだろう（ただし，この方法での解析の可能性についてはいまだ十分検討していない．また一階述語論理記述への自動翻訳は未実現である）．
- 静的解析の目標を問題レベルの生存性のみに集中することができる（その他の正当性については，原問題との一致性を除いて，FNL 言語システムが保証しているので）．
- FNL では問題仕様を（抽象度についても）原問題に近い形で記述できるので，各主体の行動本性の記述が，原問題に照らして適切であるか否か（たとえば，過剰に具体化していないか，あるいは，必然性があるかなど）を点検しやすい．先の (2-1) で述べたデッドロックの例のように，問題レベルの生存性に関する不具合は，そもそも主体の行動本性記述が上記の意味で適切でないところから生じるものと見ることもできる．記述の適切さの観点で各主体の行動本性記述を個別に点検することにより（主体たちの行動の並行合成をすることなく），問題レベルの生存性に関する不具合を回避できることも多い．

(2-4)　問題レベルの公平性（fairness property）の保証

FNL では，公平性は，どの主体も無限回の行為機会があるという条件として，FNL 言語システムが通常の OS のプロセスあるいはスレッドのスケジューリング機構に依存しつつ保証する．なお，FNL 意味論ではこの意味の公平性は，ある主体が行為主体関数の値として有限回しか現れないということはない，という要請として定義する．

(3)　実現レベルの正当性の保証

実現レベルの正当性はすべて FNL 言語システムが保証する．以下個々に見てみる．

(3-1)　問題記述と実現の一致性の保証

FNL 言語システムが問題記述からプログラムを自動生成するので，これは保証される．

(3-2)　実現レベルの安全性の保証

これはいわゆるプログラムエラー（たとえば NULL ポインタ参照など）がないという条件であるが，FNL 言語システムがプログラムを自動生成し，FNL 言語システムの実行系がそれを実行するので，この条件は保証される．

(3-3)　実現レベルの生存性の保証

FNL の逐次行為「P => Q」はプログラム言語では「P; Q」（「;」は通常のプログラム言語の連接子）に変換されるので，いずれ Q が実行されることはプログラム言語が保証する．

*1 おおがかりなバックトラックおよびロールバック機構（つまり，やりなおし機構）を仮定すれば，（問題記述が無矛盾であるかぎり）実行時に問題レベルの生存性を保証することも不可能ではないが，それはかならずしも現実的ではないのでここでは除外する．

© 2011 Information Processing Society of Japan

また，プログラム実行時に使用する lock 命令および unlock 命令によって生じうるデッドロックなども FNL 言語システムの責任でそれを回避する．

(3-4)　実現レベルの公平性の保証

通常の OS のスレッドあるいはプロセスのスケジューリング機構はこれをみたしている（FNL 言語システムは FNL における 1 つの主体を 1 つのプロセスあるいはスレッドとして動作させる）．

5.4　既存手法および言語との考え方の相違点

技術的内容とは別にそのねらいや考え方について，FNL には既存の手法および言語のそれと異なっているところがある．本節では，それらのうち特に混同されやすい点についてあらためてまとめておく．

(1)　既存の形式仕様言語とのねらいの違い

既存の多くの形式仕様言語が，すでに設計済みのシステム（ソフトウェアシステムとは限らない）を事後に検証することを想定しているのに対し，FNL は，システム（特にソフトウェアシステム）を新規に作成し同時にその正しさを保証したい場合に用いられることを主に想定している．また，既存の形式仕様言語が対象システムの厳格な記述を目標としているのに対し，FNL はさらにその自然な記述にも力点を置いている．

(2)　プログラム言語との違い

問題純粋性（実現隠蔽性）と推論性が FNL とプログラム言語との本質的な違いである．FNL でも厳格に定義された語彙（特にプログラム言語にも類似のものがある時相詞や論理詞）を用いて正確に対象を記述することを見て，ただちにプログラム言語と同等視してしまう技術者も多いが，それは，ここで述べた両者の本質的違いを無視した誤解というべきである．たとえば図 10 銀行口座問題の記述における「the Account:a's balance := a's balance − the Number:n」という FNL における最も単純な 1 つの文が，検索処理などを含む多数行からなるプログラムに展開され，かつ記述者が直接明示していない同時実行制御や待機制御をも論理的に含意していたことを思い起こされるとよい．一方，FNL の記述は手続き的に読める（もちろん宣言的にも読める）点はプログラム言語との類似点である．これは一般技術者にとっての親近性になるとともに，すでにプログラム言語で構築されたシステムの仕様を粗描するときなどに使いやすいという特徴にもなっている．

(3)　オブジェクト指向との違い

背後にある存在論が似通っているが，オブジェクト指向の主眼は，問題世界の記述ではなく，あくまでソフトウェア設計および構築の方法論にある[41]．存在論はそのための隠喩の役割を果たすものである．一方，FNL では問題世界の記述が最大の関心事であり，その存在論は FNL による世界記述の基本的枠組みという位置づけを持つ．たとえば，オブジェクト指向における記述の模範型を示すデザインパターン[42] では Iterator（オブジェクト群の線形的走査）や Visitor（複合オブジェクトの構造的走査）などのクラス（FNL の存在論では実体に相当する）を導入することが奨められるが，FNL ではこれらは実現アルゴリズムに関する概念と位置づけられ，問題記述の中に実体として登場することはない．このように，FNL では，ソフトウェア設計および構築は言語システムが自動的に行い，一般開発者にはそれを隠蔽すべきものであるという位置づけに徹している点が，オブジェクト指向との大きな相違点である．

(4)　UML との違い

FNL では，問題記述のみが開発者の作るべき主文書である．UML ではクラス図，状態遷移図，およびシーケンス図などがその主文書になるが，先に述べたように，FNL では，ソフトウェア設計および構築は言語システムが自動的に行うので，それらの設計用文書を人が書く必要がない．実際，それらを自動生成できるだけの情報は問題記述の中に含まれている．たとえば先に FNL の実行方式を述べたが，そこでは問題記述からプログラムが使用するデータ構造を自動生成していた．これはすでにクラス図を自動生成したことに相当する．また，問題記述からプログラムのアルゴリズムを自動生成しているということは，実質的に状態遷移図を生成していることに相当する．

(5)　ソフトウェア開発方法論

FNL はソフトウェア開発の各種技法およびツールがその中に凝縮された高位のミドルウェア層の具体的提案になっている（たとえば 5.1.2 項で述べたように，トランザクションや排他制御などもソフトウェア開発者は意識しなくてよくなる）．これにより現行のソフトウェア開発工程の中の設計工程からテスト工程は大幅に省略される．さらに広くはソフトウェア開発者が FNL という 1 つのインタフェースを通じてソフトウェア工学の諸成果をおのずと享受できる道が開ける．

なお，通常どおりに，実行効率の問題は漸進的課題として永く残るが，本稿で提案したシナリオによれば，実行効率向上のための技術が FNL 言語システムの中に劣化なく蓄積されるので，巨視的観点ではむしろ有利であろうと考えている．

5.5　形式仕様の将来課題との関係

文献 43) では，形式仕様についての将来課題として 16 項目をあげている．そのうち主要 8 項目については，2 章の FNL の設計方針でみずから課した要請とも共通するものであり，

© 2011 Information Processing Society of Japan

その意味で，FNL はそこであげられた形式仕様の将来課題に対して 1 つの具体的解答案を示したと考えている．以下それについて見てみる．

① Constructiveness：高位仕様から実行仕様の構成という課題と解釈すると，FNL の問題純粋性および実行可能性がこれにあたる．

② Comparative analysis：仕様記述の良否への関心の移動という課題と解釈すると，FNL の問題純粋性がこれにあたる．FNL では問題（仕様）記述と実現とを厳しく区別したことにより仕様記述の良否についての議論がしやすくなると思われる．

③ Integration：ゴール記述から設計へ，ならびに非形式的記述から形式的記述へ，という課題と解釈すると，FNL の問題純粋性，自然語指向，および実行可能性がこれにあたる．

④ Higher level abstraction：プログラム指向から問題指向へという課題と解釈すると，FNL の存在論および問題純粋性がこれにあたる．

⑤ Richer structuring mechanisms：問題指向の構造化記述という課題と解釈すると，FNL の存在論および問題純粋性がこれにあたる．

⑥ Separation of concerns：事実（descriptive）と命令（prescriptive）の分離という課題と解釈すると，FNL における事態と行為の区別，ならびに恒常的制約条件の記述と主体の行動目標の記述の分離がこれにあたる．

⑦ Lightweight techniques：数学の隠蔽という課題と解釈すると，FNL の自然語指向がこれにあたる．

⑧ Multiparadigm specification：種々の形式仕様言語とその意味論の統合という課題と解釈すると，FNL の統合性および形式性がこれにあたる（付録 A1 に世界の静的構造，動的振舞い，および並行性を統合した FNL の形式的意味論を示した）．

なお，文献 43) であげられた課題のうち，Extended scope（非機能要求記述への拡張）については，FNL は機能要求記述に集中するため意図して考究の対象としなかった．非機能要求の記述については当面自然言語に頼むのが妥当であると考えている．

6. おわりに

ソフトウェアの問題記述のための実行可能な形式化自然語（FNL）を提案した．これにより，既存の形式仕様言語がかかえる現場適用上の諸困難を打開する「もう 1 つの」形式仕様言語の可能性を具体的に示した．

本研究の貢献は以下のように要約される．

① 記述性：一般のソフトウェア技術者に使用されるために形式仕様言語がみたすべき基準を明示し，それらを満足する言語仕様を FNL として具体的に設計し提示した．

② 形式意味論：自然な存在論を明示し，それに基づき問題世界の静的構造，動的振舞い，および並行性を統合的に表現する形式意味論を構築した．それは多主体並行型の時区間述語論理になっており，そこには，主体概念や単位時区間行為の概念など，標準的論理では明示されていなかった有用な概念が他の諸概念と有機的に結び付いた形で含まれている．

③ 実行可能性：FNL 言語システムを実装した．これにより，問題純粋な（実現を隠蔽した）問題記述から実行可能な正しいプログラムを自動的に導出可能であることを実証した．しかし，効率の良いプログラムを自動導出することは今後さらに検討すべき大きな課題である．

④ ソフトウェア検証：ソフトウェア検証に関する新規の概念を提案した：(a) 記述 $\supseteq$（実行 + 検証），(b) 内包検証，(c) 正当性 = 問題レベルの正当性 + 実現レベルの正当性．ただし，実行効率のような未解決の大きな課題があり，記述 $\supseteq$（実行 + 検証）を達成したことを無条件に主張するものではもちろんない．

⑤ ソフトウェア開発プロセス：人間は問題（要求）の記述に集中し，設計，プログラミング，および検証は自動化するというソフトウェア開発過程の実現シナリオを示した．ソフトウェア工学上の多くの概念（たとえば，状態モデル，オブジェクト指向，並行制御など）も FNL 意味論の中に隠蔽される．

一方，残存課題も多い．そのうち主要な課題は以下のとおりである．

① 記述性に関して：問題記述の良否基準のさらなる考察．5.3.1 項で述べた「原問題を直接的かつ良形に記述する」とはどういうことであるかをさらに考察する必要がある．これは内包検証（外延検証における状態爆発の困難克服の一方策）において最も重要な課題である．

② 形式意味論に関して：(a) 主体概念を用いての非単調推論の拡張とその厳密な定式化．その他 (b) 線形論理[44] やアクション言語[45] などとの比較も興味深い．

③ 実行可能性に関して：(a) 恒常的制約条件のプログラム組み込み，(b) 同時実行の and，および (c) 計画挙動について，極力シンプルな実行方式を考案したい．

④ 実行性能に関して：本研究では実行性能についてはまったく考慮が及んでいない．実行性能に関しては 5.2.2 項で述べた種々の課題がある．それぞれ関連する研究分野の成果と連携しつつ長期にわたる漸進的課題として取り組む必要がある．

新規の言語であるので，本稿では，まずその全体像と意義を示すことに注力した．本稿で提示した本言語の全体像から，そこになんらかの意義が認められるなら，稿を改めてさらに細部および応用の議論に向かいたい．

© 2011 Information Processing Society of Japan

参　考　文　献

1) Spivey, J.M.: *The Z Notation: A Reference Manual*, *2nd edition*, Prentice-Hall (1992).
2) Jones, C.B.: *Systematic Software Development using VDM*, *2nd edition*, Prentice-Hall (1990).
3) Goguen, J.A., Winkler, T., Meseguer, J., Futatsugi, K. and Jouannaud, J.P.: Introducing OBJ, Technical Report SRI-CSL-92-03, Computer Science Laboratory, SRI International (1992).
4) Hoare, C.A.R.: *Communicating Sequential Processes*, Prentice-Hall (1985).
5) 谷津弘一：形式的方法概観，*UNISYS TECHNOLOGY REVIEW*, No.60, pp.67–84 (1999).
6) 中島　震：ソフトウェアの道具としての形式手法，NII Technical Report, NII-2007-007J (2007).
7) 柳生孝昭：要求仕様の形式的記述，*UNISYS TECHNOLOGY REVIEW*, No.60, pp.57–66 (1999).
8) 佐原　伸：形式手法の技術講座，ソフト・リサーチ・センター (2008).
9) 佐伯元司，米崎直樹，榎本　肇：自然言語の語彙分割による形式的仕様記述，情報処理学会論文誌，Vol.25, No.2, pp.204–215 (1984).
10) 日野克重：日常的世界観にもとづく実行可能な仕様記述言語 NAIVE，日本ソフトウェア科学会第 8 回論文集，pp.333–336 (1991).
11) Darimont, R. and van Lamsweerde, A.: Formal Refinement Patterns for Goal-Driven Requirements Elaboration, *SIGSOFT'96*, ACM (1996).
12) 岡田光弘：オントロジー応用のための方法論の考察と展望，人工知能学会誌，Vol.17, No.5, pp.604–613 (2002).
13) 溝口理一郎：オントロジー工学（知の科学），オーム社 (2005).
14) 日野克重：実世界モデルに基づく言語 NAIVE の論理体系（再論），情報処理学会研究報告，pp.1–8 (1992).
15) Gallin, D.: *Intensional and Higher-Order Modal Logic*, North-Holland (1975).
16) Dowty, D.R., Peter, S. and Wall, R.E.: *Introduction to Montague Semantics*, D.Reidel, Dordrecht, Germany (1981).
17) Andrews, P.B.: *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*, *2nd Edition*, Kluwer Academic Publishers (2002).
18) 佐伯元司：非単調命題時間論理とその形式的仕様記述への応用，情報処理学会論文誌，Vol.28, No.6, pp.547–557 (1987).
19) 佐藤　健，横田治夫，北上　始：最小変化ーフレーム問題への 1 アプローチ，人工知能全国大会（第 1 回）論文集 (1987).
20) 荒木啓二郎，張　漢明：プログラム仕様記述論，pp.168–199, オーム社 (2002).
21) 大堀　淳：プログラミング言語の基礎理論，共立出版 (1997).
22) Hoare, C.A.R.: Communicating Sequential Processes, *C. ACM*, Vol.21, No.8, pp.666–677 (1978).
23) 清野貴博，緒方和博，二木厚吉：代数仕様言語 CafeOBJ による鉄道信号システムの記述と検証，FOSE99 (1999).
24) Rose, G.A.: Object-Z, Object Orientation in Z, *Workshops in Computing*, Springer (1992).
25) Hino, K.: PROGRAMMING LANGUAGE PROCESSING SYSTEM WITH PROGRAM TRANSLATION PERFORMED BY TERM REWRITING WITH PATTERN MATCHING, United State Patent 5-530-863 (1996).
26) Fuchs, N.E. and Schwitter, R.: Specifying Logic Programs in Controlled Natural Language, *CLNLP95, WORKSHOP ON COMPUTATIONAL LOGIC FOR NATURAL LANGUAGE PROCESSING* (1995).
27) Fuchs, N.E., Schwertel, U. and Schwitter, R.: *Attempto Controlled English (ACE) Language Manual Version 3.0*, Institut fur Infomatik der Universtat Zurich (1999).
28) Berry, D.M., Kamsties, E. and Krieger, M.M.: From Contract Drafting to Software Specification: Linguistic Sources of Ambiguity, Technical Report, Computer Science Dept., Univ. of Waterloo (2003).
29) Jones, S.P.: Beautiful concurrency, "*Beautiful Code*", Wilson, G. (Ed.), O'Reilly (2007).
30) Bresciani, P., et al.: Tropos: An Agent-Oriented Software Development Methodology, *Autonomous Agents and Multi-Agent Sytems*, Vol.8, No.3, pp.203–236 (2004).
31) Fischer, C.: CSP-OZ: A combination of Object-Z and CSP, Technical Report, Fachbereich Informatik, University of Oldenburg (1997).
32) Butler, M.J.: csp2B: A practical approach to combining CSP and B, *World Congress on Formal Methods*, pp.490–508, Springer (1999).
33) 来間啓伸：B メソッドによる形式仕様記述，pp.112–114, 近代科学社 (2007).
34) Jackson, D. and Jackson, M.: *Problem Decomposition for Reuse* (1995).
35) Jackson, D.: Alloy: A Lightweight Object Modelling Notation, *ACM Trans. SE&M*, Vol.11, No.2, p.282 (2002).
36) 萩谷昌己，西崎真也：論理と計算のしくみ，岩波書店 (2007).
37) 本位田真一（監修），磯部祥尚，櫻庭健年，田口研治，田原康之，粂野文洋，田中　譲：ソフトウェア科学基礎，近代科学社 (2007).
38) 湯浅能史，武山　誠，関澤俊弦，田辺良則，高橋孝一：自動証明系と対話型証明支援系の連携によるポインタ操作プログラムの検証について，日本ソフトウェア科学会第 23 回大会論文集 1，pp.1–7 (2006).
39) Clarke, E.M., Grumberg, O. and Peled, D.: *Model Checking*, The MIT Press (2000).
40) Chandy, K.M. and Misra, J.: The Drinking Philosophers Problem, *ACM Trans.*

© 2011 Information Processing Society of Japan

Prog. Lang. Syst., Vol.6, No.4, pp.632–646 (1984).
41) Jackson, M.: *Problem Frames: Analyzing & Structuring Software Development Problems*, ACM Press.
42) Gamma, E., Helm, R., Johnson, R. and Vissides, J.: *Design Patterns Elements of Reusable Object-Oriented Software*, Addison Wesley Longman, Inc. (1995). 本位田真一，吉田和樹（監訳）：オブジェクト指向における再利用のためのデザインパターン，ソフトバンクパブリッシング (1999).
43) van Lamsweerde, A.: Formal Specification: a Roadmap, International Conference on Software Engineering, *Proc. Conference on The Future of Software Engineering*, pp.147–159 (2000).
44) Girard, J.Y.: Linear Logic, *Theoretical Computer Science*, Vol.50, pp.1–102 (1987).
45) 鍋島英知：アクション言語を用いた状態変化の表現と推論に関する研究，神戸大学大学院自然科学研究科博士論文 (2001).
46) 二村良彦，雨宮真人，山崎利治，淵　一博：新しいプログラミング・パラダイムによる共通問題の設計，情報処理，Vol.26, No.5, pp.458–459 (1985).
47) Problem Set for the Fourth International Workshop on Software Specification and Design, *4TH INTERNATIONAL WORKSHOP ON SOFTWARE SPECIFICATION AND DESIGN*, IEEE COMPUTER SOCIETY PRESS (1987).
48) 柴山悦哉，米澤明憲：並列オブジェクト指向言語，*bit*, Vol.20, No.8, pp.940–954, (1988).
49) 渡辺慎哉，原田康徳，三谷和史，宮本衛市：場とイベントによる並列計算モデル：Kamui 88，コンピュータソフトウェア，Vol.6, No.1, pp.41–55 (1989).
50) 黒川利明：Prolog の作法，岩波書店 (1985).
51) Karam, G.M., Stanczyk, C.M. and Bond, G.W.: Critical Races in Ada Programs, *IEEE Trans. Softw. Eng.*, Vol.15, No.11, pp.1471–1480 (1998).
52) 榎本　肇（編）：ソフトウェア工学ハンドブック，オーム社 (1986).
53) 山崎利治：プログラムの設計，共立出版 (1990).
54) Lambert, K.: *Free Logic: Selected Essays*, Cambridge Univ. Press (2002).

付　　録

A1　言語 FNL の構文論・意味論・語用論

FNL の形式的言語仕様は，内包論理[15),16)] に準拠し，その上に FNL 固有の存在論を反映した多主体並行型の時区間述語論理というべきものになっている．

(1)　構文論

YA.　型

YA1.　型の集合

型の集合 Ty は，以下をみたす最小集合である．

①　基本型

e（実体），n (自然数) ，k（文字列），g（使者），t（事態），p（行為）$\in$ Ty

②　$e_1, \sim, e_n \in$ Ty

③　u $\in$ Ty かつ v $\in$ Ty ならば <uv> $\in$ Ty

④　u $\in$ Ty ならば <su> $\in$ Ty

<注> 上記 ② の e_i は，問題ごとに記述者が使用する普通名詞のことである．
また ④ にある s は指標（index）であり，通常（可能世界意味論における）世界や時点を表す．s 自体は型ではない．

YA2.　型の階層

型の集合上では半順序関係 $\leqq$ が定義されており，以下の条件をみたしている．

①　$e_1, \sim, e_n \leqq$ e

②　t $\leqq$ p

③　x $\leqq$ u かつ v $\leqq$ y ならば <uv> $\leqq$ <xy>

YB.　整合式

YB1.　定項と変項

①　それぞれの型 u（型 g を除く）に対して，可算個の非論理定項と可算個の変項がある．

②　FNL の論理定項（基本詞）としては，実在詞，主体詞，時相詞，限定詞，および算術詞があるが，これらについては MB 整合式の意味規則であわせて示す．

③　すべての非論理定項（問題記述者が導入する語彙）は，基礎定項（実質定項）と派生定項（名目定項）とに分類される．他のなんらかの定項によって定義されている定項は派生定項とされる．基礎定項と派生定項の区別は後述の「現実的モデル」の判定時に有効になる．

YB2.　構文規則

整合式の集合 WF_u は以下をみたす最小集合である．

①　$C_u \in WF_u$　（C_u は型 u の定項）

②　$x_u \in WF_u$　（x_u は型 u の変項）

③　$A_{<uv>} \in WF_{<uv>}$，$B_u \in WF_u$ ならば $AB \in WF_v$

© 2011 Information Processing Society of Japan

④　$x_u \in WF_u$，$A_v \in WF_v$ ならば $\lambda x_u.\ A_v \in WF_{<uv>}$

⑤　$A_u \in WF_u$，$B_u \in WF_u$ ならば $[A \equiv B] \in WF_t$

⑥　$A_u \in WF_u$ ならば $\uparrow A_u \in WF_{<su>}$

⑦　$A_{<su>} \in WF_{<su>}$ ならば $\downarrow A_{<su>} \in WF_u$

(2)　意味論

MA.　解釈（モデル）

FNL の整合式は，行為主体と時区間で解釈される．つまり，たとえば行為式 p は，行為主体 a と時区間 <i, j> が与えられてはじめて，その真偽が定まる．

いずれの型に対応する解釈領域も要素として Nil を持つ．Nil とは，直感的には未定義の対象あるいは無のことである（特に真偽値領域も Nil を含むので，正確には FNL は 3 値論理であることになる）．Nil は FNL の意味規則の定義で使うが，問題記述者は Nil を直接意識する必要はない（実在詞 nil を使うことはあるが，ここでいう Nil 要素を意識する必要はない）．

MA1.　フレーム

フレーム $\{D_u\}$（$u \in Ty$）を次のように定める．

①　$D_e = E \ni Nil_e$；実体領域

②　$D_{ei} \subseteq D_e$；各種族領域

なお，$u \leqq v$ ならば，$D_u \subseteq D_v$ である．

③　$D_g = G \ni Nil_g$；使者領域

④　$D_t = \{T, F, Nil\}$；事態の真偽値領域

⑤　$D_p = \{T, F, Nil\}$；行為の真偽値領域

⑥　$D_n = N \cup \{Nil_n\}$；自然数領域

⑦　$D_k = K \ni Nil_k$；文字列領域

⑧　$D_{<uv>} = \{f \mid f\colon D_u \to D_v\}$；$D_u$ から D_v への関数領域

⑨　$D_{<su>} = \{f \mid f\colon S \to D_u\}$；$D_u$ の内包化領域．

ここで，$S = \{<i, j> \mid i, j \in N,\ i \leqq j\}$（時区間全体の集合）

MA2.　解釈

解釈 M とは，順序対 $<\{D_u\}, h, m>$ のことである．ここで，

$\{D_u\}$：MA1 で述べたフレーム

h：行為主体関数．各時区間 <i, j>（$\in N \times N$）に対して，$D_e \cup D_g$ の部分集合を割り当てる関数．FNL における多主体の並行動作を表現するものである（**図 17**）．

<注> この行為主体関数の定義は，1 つの単位時区間で複数の主体が同時に行為しうるいわゆる「真の並列実行」も想定していることを意味する．割り当てる $D_e \cup D_g$ の部分集合の濃度を 1 に限定すれば，インタリーブ的並行動作のみを許すことになる．なお，FNL の現実行方式では，インタリーブ的並行動作のみを許している．

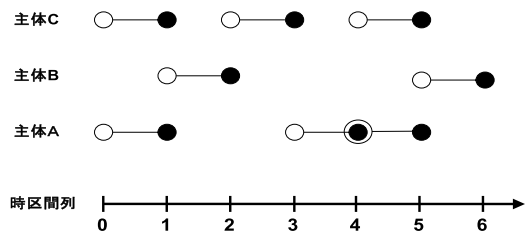

図 17　FNL における多主体の並行動作
Fig. 17　Subjects' concurrency over intervals.

m：意味関数．論理定項 existent および nil ならびに各非論理定項 C_u に対して $D_{<su>}$ の要素を割り当てる関数．正解釈（モデル）は，以降の MB の意味規則をみたす．

MB.　整合式についての意味規則

任意の整合式 W に対し，解釈 M，a，<i, j> $\in$ S，および変項割当て関数 α に対して意味値を与える関数を付値関数 V とよぶ．ここで，変項割当て関数 α は，各変項 x_u に D_u の要素を対応させる関数である．なお，以降，「V〔M, a, <i, j>, α, W〕」を「Va, i, j, α〔W〕」と略記する．<注> たとえば「Va, i, j, α〔W_p〕= T」は，直感的には，「行為主体 a が時区間 <i, j> において W を真にする（実行する）」と読めばよい．

以下，付値関数 V がみたすべき条件を示す．

<u>非論理定項（記述者が導入した語彙；関係詞や属性詞など）と変項（指示詞）</u>

①a　任意の定項 C_u に対して，

$$Va, i, j, \alpha〔C_u〕 = \begin{cases} (m(C_u))(<i, j>), & \text{if } a \in h(i, j) \\ Nil_u, & \text{otherwise.} \end{cases}$$

<注 1>どんな定項 C も，行為している主体のみが参照･更新できることを意味している．

<注 2>この規則は，たとえば，「Va, i, j, α〔C_p〕= T」から「$a \in h(i, j)$」および「$(m(C_p))(<i, j>) = T$」を導出するときなどに使う．

①b　型 e，型 n，および型 k の任意の定項 C に対して，$(m(C_e))(<i, j>) = (m(C_e))(<0, 1>)$

© 2011 Information Processing Society of Japan

$(m(C_n))(<i,j>) = (m(C_n))(<0,1>)$

$(m(C_k))(<i,j>) = (m(C_k))(<0,1>)$

＜注＞これらは，実体の固有名詞，数詞，文字列詞の意味（指示対象）は，時間や行為主体を通じて不変であるという存在論を反映したものである．

② 変項 x_u に対して，

$Va, i, j, \alpha〔x_u〕 = \alpha(x_u)$

＜注＞変項は，自然語における指示詞と同様に使われる．プログラム言語のように代入を許すものではない．

内包論理の整合式

③ $Va, i, j, \alpha〔P_{<uv>}Q_u〕 =$

$Va, i, j, \alpha〔P_{<uv>}〕(Va, i, j, \alpha〔Q_u〕)$

④ $Va, i, j, \alpha〔\lambda x_u\ (P_v)〕 =$

$\lambda x_u\{Va, i, j, \alpha〔P_v〕\}$

⑤ $Va, i, j, \alpha〔P_u \equiv Q_u〕 = T$, iff

$Va, i, j, \alpha〔P_u〕 = Va, i, j, \alpha〔Q_u〕$

⑥ $Va, i, j, \alpha〔\uparrow P_u〕 =$

$\lambda<i,j>\{Va, i, j, \alpha〔P_u〕\}$

⑦ $Va, i, j, \alpha〔\downarrow P_{<su>}〕 =$

$(Va, i, j, \alpha〔P_{<su>}〕)(<i,j>)$

事態詞

⑧a $Va, i, j, \alpha〔true_p〕 = T$

$Va, i, j, \alpha〔false_p〕 = F$

＜注＞ true，false は，通常どおりの意味である．

特に，「$\uparrow true \equiv \uparrow P_p$」は，P が恒常的制約条件であるという定義式になる．

⑧b $Va, i, j, \alpha〔P_t〕 = Va, i, k, \alpha〔P_t〕$

＜注＞事態式の真偽は，時区間の先頭時点での真偽で決まるという意味である．一方，行為式の意味は，あくまで行為主体と時区間で決まる．

実在詞

⑨a $Va, i, j, \alpha〔nil_{<et>}A_e〕 = T$，iff

$Va, i, j, \alpha〔existent_{<et>}P_e〕 = F$，iff

$Va, i, j, \alpha〔A_e \equiv Nil_e〕 = T$

＜注＞ existent とは特定の実体が実在しているということである．nil とはその実体が実在していないということである．なお，後述の ◎<pp>（次時点詞）とあわせて用いることによって，実体を生成する，あるいは，実体を消滅させるという意味を表すことができる．

⑨b $Va, 0, 0, \alpha〔existent_{<et>}A_e〕 = F$

＜注＞これは，システム起動の初期にはどんな実体も実在していないという存在論を反映したものである．

⑨c $Va, i, j, \alpha〔P_{<eu>}Nil_e〕 = Nil_u$

＜注＞これは，実在していないものはどんな属性や関係も持たないし，どんな操作も無操作になるという存在論を反映したものである（デザインパターンにおけるヌルオブジェクトの考えかたに相当する）．

主体詞

⑩ $Va, i, j, \alpha〔@x_A〕 = T$，iff

$\alpha(x_A) = a$，かつ

$i \leqq k \leqq j$ なるすべての k について

$((m(existent))k)a = T$

＜注＞これは，実体はそれが実在している限り，その行動本性（$@x_A$）に従って行動するという存在論を反映したものである．

時相詞と論理詞

⑪a $Va, i, j, \alpha〔◎\ P_p〕 = T$ ならば

$Va, i+1, j, \alpha〔P_p〕 = T$

＜注＞ ◎ P_p ：次時点行為を表す．すなわち，次時点で P を真にする．

⑪b $Va, i, j, \alpha〔P_p => Q_p〕 = T$, iff

$Va, i, m, \alpha〔P_p〕 = T$　　かつ

$Va, n, j, \alpha〔Q_p〕 = T$

where $i < m < n < j$

＜注＞ $P_p => Q_p$：逐次行為．P を実行して，その後 Q を実行する（◎と異なり，P の実行と Q の実行の間に他の主体の行為が割り込む可能性がある）．

⑪c $Va, i, j, \alpha〔not\ (P_p)〕 = T$，iff

$Va, i, j, \alpha〔P_p〕 = F.$

$Va, i, j, \alpha〔not(P_p)〕 = F$，iff

© 2011 Information Processing Society of Japan

$\mathrm{Va,i,j,}\alpha\langle \mathrm{P_p}\rangle = \mathrm{T}.$

$\mathrm{Va,i,j,}\alpha\langle \mathrm{not(P_p)}\rangle = \mathrm{Nil}$，iff

$\mathrm{Va,i,j,}\alpha\langle \mathrm{P_p}\rangle = \mathrm{Nil}.$

⑪d　$\mathrm{Va,i,j,}\alpha\langle \mathrm{P_p\ and\ Q_p}\rangle =$

$\min\{\mathrm{Va,i,j,}\alpha\langle \mathrm{P_p}\rangle; \mathrm{Va,i,j,}\alpha\langle \mathrm{Q_p}\rangle\}$

⑪e　$\mathrm{Va,i,j,}\alpha\langle \mathrm{P_p\ or\ Q_p}\rangle =$

$\max\{\mathrm{Va,i,j,}\alpha\langle \mathrm{P_p}\rangle; \mathrm{Va,i,j,}\alpha\langle \mathrm{Q_p}\rangle\}$

<注> and，or，if：事態式について使用されたときは通常の意味と同じである．行為式について使用されたときは，and は同時実行，or は選択実行の意味を持つ．

<注>同一行為の同時複数実行不可能性の意味規則：P と Q が同一の基礎行為である場合は，「$\mathrm{P_p\ and\ Q_p}$」は無意味（同時には実行不可能）であるとしてつねに false と見なす．ここで同一の基礎行為とは，基礎行為の名前とそれに関与する実体（基礎行為のパラメータ）がすべて一致する基礎行為のことである．この意味規則は，実体の資源性を扱うときに使われる．

⑪f　$\mathrm{Va,i,j,}\alpha\langle \mathrm{if\ P_p\ Q_p}\rangle = \mathrm{Va,i,j,}\alpha\langle \mathrm{P_p\ ->\ Q_p}\rangle =$

$\mathrm{Va,i,j,}\alpha\langle \mathrm{not(P_p)\ or\ Q_p}\rangle$

⑪g　$\mathrm{Va,i,j,}\alpha\langle \mathrm{when\ P_p\ Q_p}\rangle = \mathrm{T}$, iff

$\mathrm{Va,i,j,}\alpha\langle(\mathrm{if\ P_p\ Q_p})\ \mathrm{and}$

$(\mathrm{if\ not(P_p)}(◎\ (\mathrm{when\ P_p\ Q_p})\rangle = \mathrm{T}$

<注> $\mathrm{when\ P_p\ Q_p}$：待機行為．P が真になるのを待って Q を実行する．

⑪h　$\mathrm{Va,i,j,}\alpha\langle \mathrm{while\ P_p\ Q_p}\rangle = \mathrm{T}$　iff

$\mathrm{Va,i,j,}\alpha\langle(\mathrm{if\ P_p\ (Q_p\ =>\ while\ P_p\ Q_p)})\rangle = \mathrm{T}$

<注> $\mathrm{while\ P_p\ Q_p}$：継続行為．P が真である間，Q の実行を継続し，P が偽になるとその実行をやめる．

限定詞

⑫a　$\mathrm{Va,i,j,}\alpha\langle \Pi_{<<\mathrm{ep}><\mathrm{pp}>>}\{\lambda \mathrm{x_A(P_p)}\}\ \mathrm{Q_p}\rangle = \mathrm{T}$, iff

$\mathrm{x_A}$ に対する割当てを除いて α と同じすべての変項割当て関数 α' について，

$\mathrm{Va,i,j,}\alpha'\langle(\mathrm{existent(x_A)\ and\ P_p})\ \mathrm{->\ Q_p})\rangle = \mathrm{T}$

<注>「for all」の形式的意味である．⑪ の and（同時実行）とあいまって，すべての A について「同時に」Q を実行するという意味になる．

⑫b　$\mathrm{Va,i,j,}\alpha\langle \Sigma_{<<\mathrm{ep}><\mathrm{pp}>>}\{\lambda \mathrm{x_A(P_p)}\}\ \mathrm{Q_p}\rangle = \mathrm{T}$, iff

$\mathrm{x_A}$ に対する割当てを除いて α と同じある変項割当て関数 α' について，

$\mathrm{Va,i,j,}\alpha'\langle(\mathrm{existent(x_A)\ and\ P_p})\ \mathrm{and\ Q_p}\rangle = \mathrm{T}$

<注>「for some」の形式的意味である．

⑫c　$\mathrm{Va,i,j,}\alpha\langle \Theta_{<<\mathrm{ep}><\mathrm{n}<\mathrm{pp}>>}\mathrm{N_n}\{\lambda \mathrm{x_A(P_p)}\}\ \mathrm{Q_p}\rangle = \mathrm{T}$, iff

$\mathrm{x_A}$ の割当て値が異なる以外は α とまったく同じで，かつ相互に異なる N 個以上の割当て関数 α_1,〜, α_k〜, α_N について，

$\mathrm{Va,i,j,}\alpha_\mathrm{k}\langle(\mathrm{existent(x_A)\ and\ P_p})\ \mathrm{and\ Q_p}\rangle = \mathrm{T}$

<注>「for many」の形式的意味である．

⑫d　$\mathrm{Va,i,j,}\alpha\langle \mathrm{I}_{<<\mathrm{ep}><\mathrm{pp}>>}\{\lambda \mathrm{x_A(P_p)}\}\ \mathrm{Q_p}\rangle = \mathrm{T}$, iff

$\mathrm{x_A}$ の割当て値のほかは α とまったく同じ唯一の割当て関数 α' があって

$\mathrm{Va,i,j,}\alpha'\langle(\mathrm{existent(x_A)\ and\ P_p})\ \mathrm{and\ Q_p}\rangle = \mathrm{T}$

<注>「for the」の形式的意味である．

算術詞

⑬a　$\mathrm{Va,i,j,}\alpha\langle\#\{\lambda \mathrm{x_A(P_t)}\}\rangle =$

$\mathrm{Va,i,j,}\alpha'\langle\mathrm{existent(x_A)\ and\ P_t}\rangle = \mathrm{T}$ となるような α' の個数．ここで，α' は，$\mathrm{x_A}$ に対する割当てを除いて α と同じある変項への値割当て関数．

<注> $\#\{\lambda \mathrm{x_A(P_t)}\}$：P であるような A の個数

⑬b　$\mathrm{Va,i,j,}\alpha\langle\mathrm{sum}\ \{\lambda \mathrm{x_A(P_t)}\}\ \mathrm{Q_n}\rangle =$

$\mathrm{Va,i,j,}\alpha'\langle\mathrm{existent(x_A)\ and\ P_t}\rangle = \mathrm{T}$ となるようなすべての α' についての $\mathrm{Va,i,j,}\alpha'\langle \mathrm{Q_n}\rangle$ の総和．ここで，α' は，$\mathrm{x_A}$ に対する割当てを除いて α と同じある変項への値割当て関数．

<注> $\mathrm{sum}\ \{\lambda \mathrm{x_A(P_t)}\}\ \mathrm{Q_n}$：P であるようなすべての A についての Q の総計

⑬c　（その他 +，∗，=，< などの通常の算術演算子や数字は通常の意味を持つ）

(3)　語用論（仕様，操作，実行，現実的モデル）

FNL においては，（システムの）仕様，操作，および実行ということをそれぞれ以下のように形式的に定義する．

P.　語用規則

① 仕様：仕様とは，次の形の型 t の整合式である．

$(\uparrow \mathrm{A_u} \equiv \uparrow \mathrm{B_u})\ \mathrm{and} \cdots \mathrm{and}\ (\uparrow \mathrm{C_v} \equiv \uparrow \mathrm{D_v})$

ここで，↑ は内包化演算子であり，u および v は型を表す．

② システムの起動：システムの起動とは，システムの仕様を真にすることである．

© 2011 Information Processing Society of Japan

③ システムの操作：操作コマンド A_p を投入するとは，仕様内に定義されている世界外操作 A_p を真にすることである．

④ システムの実行：システムの実行とは，1 つの仕様と複数の操作によって課せられる制約を満足するモデルのうちの「現実的モデル」（後述）を構成することである．

⑤ 現実的モデル：主体たちの実行系列（いわゆる実行スケジュール；FNL 意味論では行為主体関数がそれに対応する）ごとにそれぞれ複数のモデルがありうるが，それら各実行系列でのモデルのうち，「適切な変化」を継続しているモデルのこと．適切な変化とは以下の 2 つの条件をみたした変化のことである．

（i）極小変化性：変化が極小であること（モデル構成上必要でない変化は起きないこと．「極小」となっているのは，変化の非決定性を考慮したものである）．

（ii）許容変化性：各主体の行為が起こす変化が，その主体の属性（主体の意思の優先度や主体の計画性など）に照らして許容される変化であること（たとえば，他主体が成立させた事態を，それと同等以下の意思優先度の主体が恒常的制約条件を順守するために不成立にすることは許容されない）．

ここで変化とは，時点 i から時点 i + 1 への推移における基礎定項たち（ただし行為定項を除く）の意味変化のことであり，その大小関係は基本的には集合としての包含関係で定義される．ただし，基礎定項 existent の意味変化は無限大と評価するなどの考慮がされる．

現実的モデルの直感的理解は容易であるが，その厳密な定式化には非単調論理[18),19)] をさらに拡張する必要がある．それは今後の課題である．

A2　読解性の実験で使用した記述例題

OCL記述[35]

```
1: Person
2: self.pets
3:   -> select (d | d.oclKindOf ( Dog) )
4:   -> forall ( d | not ( self.children
5:   -> union ( self.friends )
6:   -> collect ( q | q.name ) )
7:   -> includes ( d.name )
```

Alloy記述[35]

```
1: all p:Person | all d:p.pets & Dog |
2:   d.name!in ( p.children + p.friends ).name
```

FNL記述

```
1: for all Person:p
2:  { for all Dog:d who are p's Pets
3:    { d's Name is not any Person:x's Name
4:       where ( x is p's Children or
5:                 x is p's Friends )
6:   } }
```


© 2011 Information Processing Society of Japan

A3　プログラム自動合成の実験実績

No.	例題	概要	記述行数（FNL）	展開行数（C）
【並行処理，シミュレーション問題】				
1	迷路	迷路を自由に設定し,それを脱出する	43	465
2	哲学者の食事[4]	円卓に座った5人の哲学者がフォークを取り合いながら食事をする	37	371
3	パッケージルータ[34]	複雑な通路の中を複数の荷物がぶつかりを避けながら目的地へ流れていく	62	920
4	酒類販売店[46]	酒屋の在庫(倉庫,コンテナ,銘柄,本数)を管理し,顧客の注文に対応する	37	510
5	座席予約システム	新幹線の座席予約システム(区間単位での予約,キャンセル自動待ち,代替列車表示など)	155	1541
6	エレベータ[47]	複数台のエレベータが全体として円滑に動作	92	1047
7	ミニ通信管理	ネットワーク上での動的経路選択,流量制御	50	843
8	洗車問題[48]	ランダムに入ってくる車を複数の洗車係が公平かつ効率的に洗う,全体の管理者はいない	31	571
9	喫煙者問題	タバコ,紙,マッチの各資源を分け合いながら,複数の喫煙者が円滑に喫煙を続ける	33	420
10	分散型ジャンケン[49]	複数人がジャンケンを繰り返し,最後に一人勝ち残る,審判はいない.	49	590
11	Handshake Daisy Chain Arbiter	ループをなしたセル群が権利書をもちまわる,各セルは,その権利書を獲得したときのみ,対応するプロセスに資源使用権を与える	47	624
12	ジョブショップ問題	ランダムに注文されてくる多種類の仕事を多種類の器械で処理していく.同種類の器械でも何台かづつある	50	720
13	論理回路シミュレータ	加算器回路の設定および動作のシミュレーション	349	2550
14	自動ロック式ドア[50]	自動ロック式ドアの開閉操作の模擬	55	330
15	電話交換機	発呼,接続管理,話中処理などの交換機処理	58	600
16	飲酒する哲学者[40]	哲学者たちが何種類かの酒を競合つつ饗宴を繰りひろげる.テーブル配置なども自由	35	550
17	ガス給油所[51]	客と係とガスポンプが連動してガスを給油する,代金支払い処理もある	60	947
18	自動販売機	飲料の自動販売機のシミュレーション	115	1030
19	車輪駆動[52]	複数の車輪を順々に回転させていく	5	156
20	並行有限バッファ	有限なバッファを介して複数人が並行動作しつつ,ボールの受け渡しをする	17	250
【事務処理,データベース操作問題】				
21	図書館管理[47]	図書の貸出管理,表示サービスなどを行う	41	590
22	書店売掛管理[53]	書店の簡単な売掛管理	44	450
23	部品供給管理	部品データベースの検索処理	13	240
24	ビデオレンタルショップ	ビデオレンタルショップの貸出管理,料金計算,定期レポート,会員情報管理などを行う	79	850
【数論的,論理的問題】				
25	多元多次連立方程式	多元多次連立方程式／不等式を解く	30	361
26	整数計画問題	宣言的に記述された整数計画式を解く(最大化,最小化の表現)	20	230
27	ハノイの塔	円盤の大小関係を維持して移動する	8	180
28	福祉国家判定	動的に変化する事態をみながら,各時点での福祉国家を判定する論理問題	20	325

A4　FNL 言語システムの翻訳および論理合成例

```
all red Apple become on the Desk:d, where
    some Person is eating the Apple
```

↓　翻訳

```
"for all ( Apple:x )
        ( and ( red ( Apple:x ) )
              ( for some ( Person:y )
                         true
                         eating ( Person:y, Apple:x ) ) )
        ( becomes ( on ( Apple:x, Desk:d ) ) )"
```

```
forall(Apple:x)  {
   if ( "and ( red ( Apple:x ) )
            ( for some ( Person:y )
                       true
                       eating ( Person:y, Apple:x ) )" )
      if  ( "becomes ( on ( Apple:x, Desk:d ) )" )
         noop ;
      else
         return false ;
   else
      noop ;
}
return true ;
```

```
"and ( red ( Apple:x ) )
     ( for some ( Person:y )
                true
                eating ( Person:y, Apple:x ) )"
```

```
if ( "red ( Apple:x )" )
   if ( "for some ( Person:y )
                true
                eating ( Person:y, Apple:x )" )
      return true;
   else
      return false;
else
   return false;
```

```
"red ( Apple:x )"
if (check(1,red,Apple,x))
   return true;
else
   return false;
```

```
"for some ( Person:y )
          true
          eating ( Person:y, Apple:x ) )"
```

```
forall(Person:y)  {
   if ( true )
      if ( "eating ( Person:y, Apple:x )" )
         return true;
      else
         noop;
   else
      noop;
}
return false;
```

```
"eating ( Person:y, Apple:x )"
if (check(2,eating,Person,y,Apple,x))
   return true;
else
   return false;
```

```
"becomes ( on ( Apple:x, Desk:d ) )"
if (setrel(2,on,Apple,x,Desk,d))
  return true;
else
  return false;
```


© 2011 Information Processing Society of Japan

A5　座席予約問題の FNL 記述

```
reserve(the Member:m,the Seat:s) ==
   ( s becomes reserved-by m ) or
   ( s does not anything ) .
cancel(the Seat:s) ==
   s becomes not reserved-by any Member.
query (the Seat:s) ==
   does list some Member:m
       where ( s is reserved-by m ) .
true ==
   for all Seat:s
      { not ( s is reserved-by
                    many(2) Member ) } .
```

A6　Package Router 問題の FNL 記述

```
@ Conveyor:c ==
 while ( c is on )
   { some Package:p becomes existent  and
     p becomes at c }.
@ R-Station:r ==
   while true
      for some Package:p at r
         { p's ID := read-id (r)
           and
           p's Dest := read-dest (r)
           and
           r does flow-control }.
the Ctrl-Place:x does flow-control ==
   some Place next x becomes open.
@ Switch:s ==
   while true
       while ( no Package is at s )
           s does flow-control.
the Switch:s does flow-control ==
   for some Package:p at s's Prev
    { some Place:x becomes open, where
            ( x is next s  and
              x is reaching p's Box ) }.
@ Pipe:i ==
   while true
       i does not anything.
@ Box:b ==
   while true
     for all misrouted Package:p at b
       display ( p, b, "misrouted" ).
@ Package:p ==
   while true
       { p does slide }.
the Package:p does slide ==
   p becomes at some Place:x, where
     (x is next p's At and  x is open ).
the Package:p is at the Place:x ==
   p's At is x.
the Place:x is prev the Place:y ==
   y is next x.
the Place:x is reaching the Place:y ==
   ( x is prev y ) or
   ( x is prev some Place which is
                            reaching y ).
the Package:p is misrouted ==
   p is at some Box:b  and
   b is not p's Box.
the Package:p's Box ==
   p's Dest's Box.
Place ==
   Thru-Place or Ctrl-Place or Box.
Thru-Place ==
   Conveyor or Pipe.
Ctrl-Place ==
   R-Station or Switch.
true ==
   { for any Package:p
       not ( p is at many(2) Place ) }
   and
   { for any Switch:s
        not ( many(2) Place next s are open }
   and
  { for any Thru-Place:x
       ( all Place next x are open ) }.
```

(平成 22 年 4 月 1 日受付)
(平成 22 年 12 月 1 日採録)

© 2011 Information Processing Society of Japan

日野　克重

1972 年早稲田大学理工学部電気工学科卒業．同年富士通（株）入社．ソフトウェア開発および教育企画等の業務に従事．開発推進部部長を経て 2009 年定年退職．現在，東京大学大学院情報理工学系研究科博士課程在学中．第 4 回ソフトウェア生産における品質管理シンポジウム（1984 年日本科学技術連盟主催）最優秀論文賞受賞．上級教育士（JSEE）．

本位田真一（フェロー）

1978 年早稲田大学大学院理工学研究科修士課程修了．（株）東芝を経て 2000 年より国立情報学研究所教授，2004 年より同研究所アーキテクチャ科学研究系研究主幹を併任，現在に至る．2008 年より同研究所先端ソフトウェア工学・国際研究センター長を併任，現在に至る．2001 年より東京大学大学院情報理工学系研究科教授を兼任，現在に至る．現在，早稲田大学客員教授，英国 UCL 客員教授を兼任．2005 年度パリ第 6 大学招聘教授．工学博士（早稲田大学）．1986 年度情報処理学会論文賞受賞．日本ソフトウェア科学会理事，情報処理学会理事を歴任．ACM 日本支部会計幹事，情報処理学会フェロー，日本ソフトウェア科学会編集委員長，日本学術会議連携会員．

© 2011 Information Processing Society of Japan