(1) Algorithmic & Knowledge Based Methods : Do They "Unify"? —With Some Programme Remarks for UNU/IIST—
(アルゴリズムに基づく手法と知識に基づく手法，この2つの手法は融合するか)

Dines Bjørner(国連大学国際ソフトウェア工学研究所長)

［略歴］

Dines Bjørner。1962年デンマーク工科大学卒業。13年間IBMに勤務。1976年にデンマーク工科大学に戻り，コンピュータサイエンス学科の教授に就任。デンマーク政府情報委員会委員長やDansk Datamatic Centerの所長など，いくつかの管理ポストを経た後，現在は国連大学の国際ソフトウェア工学研究所の所長。「ウィーン開発手法」などのプログラミング手法の分野における研究者として著名。

御招待を受けて皆様の前で講演することができ，非常に光栄です。私の講演のタイトルは，もちろん，この会議の主要なテーマと私自身のバックグランドを考えて選んだものです。非常に手短かに言いますと，私の話は3つの部分から成り立っています。まず最初に，モデル指向のソフトウェア開発とアルゴリズムに基づくソフトウェア開発について，ごく一般的にお話します。次に，アルゴリズムに基づく手法と知識に基づく手法の相違点についてお話します。そして，この2つの手法を理解するための方法を提言してみようと思います。最後に，私が次に取り組もうとしている課題について少しお話します。

この講演の招待を受けたとき，会議の組織委員会の皆様，およびここにお集まりの皆様から，国家的な協力体制における研究開発の推進や創造性などに関して，国連大学の国際ソフトウェア技術研究所の戦略を知りたいとのご希望がありました。その点については，話の随所で触れて行きたいと思います。

この招待講演をお受けすることをお知らせしたときに，「日本の通産省が10年前に第五世代コンピュータプロジェクトの開始を決定したことは，知識に基づくコンピューティングシステムに関する世界規模での研究，工学，およびアプリケーションに対して，強烈かつ非常に前向きな影響を与えたと思う」ということを書きましたし，また今でもそう思っています。その1つの例として，日本は，コンピューティングの世界を，劇的に専門的かつエキサイティングなものにしてきました。

私がこのお話しをする背景は何かというと，基本的には，私自身はアルゴリズムないしモデル指向のアプローチでソフトウェア開発に取り組んでいます。ところが，このプロジェクトでここにお集まりいただいているのは，主に知識に基づくアプローチに焦点を当てた議論を行うためです。そのため，私は，同僚のJorgen Fischer Nilsson教授に，論文の執筆の支援を依頼しました。

繰り返しになりますが，最初に，アルゴリズム指向の開発の全般について，いくつかの項目の概要をごく手短かに述べます。この3つの部分については，定説をいくつかお話します。次に，この2つのアプローチの比較に触れます。ここでは，まずケーススタディを簡単に紹介します。またそれには関係なく，アルゴリズムに基づくアプローチと知識に基づくアプローチについて，私の視点から体系的な「一覧表」を示します。これは奥の深い表ではなく，ましてや科学的な表ではありません。単なる試みに過ぎません。ただし，よりよい比較を行うための手助けにはなると思います。そして，より数学的な意味で2つのアプローチを関連付けることによって，この議論を締めくくります。最後に，

私が次に取り組もうとしている課題がどんなものなのかについて概略をお話します。

アルゴリズム指向のアプローチ，つまりソフトウェア開発に対するモデル指向のアプローチでは，基本的にはソフトウェアに対する考察を始める前に，非常に包括的かつ徹底的に要求項目の作成を行います。その要求項目が完成したと確信できてはじめて，ソフトウェア開発を開始する準備が整ったことになります。

要求項目の作成は，問題領域の記述，領域の各アスペクトのモデル化，およびソフトウェアに組み込まれるアスペクトの獲得から構成されています。それが完了すると，ソフトウェア開発，仕様化，段階的な設計，および実行可能コードへと進んで行くことができます。

まず，領域のモデル化の側面についてのみ説明します。

要求項目のモデル化には，必ずしも逐次的ではない，いくつかの面(facet)やステップが関係しています。この中には，構成要素は何か，問題領域の各種の構成要素を支配する不変式は何か，発生する物事，アクション，イベントの入出力動作は何か，イベントは動作内でどのようにトレースないしリンクされるのか，また全体的に，すべてがどのように整合しているかについて理解することが含まれます。また，この中には，独立性が強いか弱いか，また独立していないかにかかわらず，個別のモデルを記述してそれを合成および関連付けることが可能なアスペクトも含まれます。

これらの作業を行なうなかで，必要があれば，モデル化対象のシステムの安全限界のアスペクト，失敗の確率，提案されたシステムの効率，およびコンピュータと人間のインタフェースなどに焦点を当てることができます。ここで述べたことは，前記の各アスペクトに適用することができます。また，以上の各アスペクトを独立にシミュレートすることもできます。しかし，これらすべてのアクティビティは問題領域の理解に関係するものであり，未だソフトウェアの理解には関係しないことに注意してください。

したがって，最初の定説は，基本的には次のようになります。我々が行おうとしていることを正確な自然言語で記述できない場合，要求項目のモデル化に関して先に挙げた各種のアスペクトを数学モデルで記述できない場合，そしてソフトウェアに入れるべき要求項目を簡潔に獲得できない場合には，その開発はほとんどまたは全く信頼できません。つまり私自身が現在取り組んでいるどの開発プロジェクトも，ほとんど信頼できないことになります。依然として，この方面では取り組むべき課題が山積みしています。

要求項目の作成が終わると，ソフトウェア開発に移ります。ソフトウェア開発とは，プログラミングとソフトウェア工学が密接に関係したものと考えています。プログラミングでは，仕様，設計およびコードを形式的なオブジェクトとして扱います。それらについて推論を行い，また属性を計算します。

次の定説を示したいと思います。ソフトウェアが果たすべき機能を記述できないままでインプリメントされてしまった場合，またソフトウェアが行うべきすべての動作とソフトウェアが構築される環境に関する前提条件を，費用のかかるインプリメンテーションの前に記述できなかった場合，またこのような機能的な仕様や動作の仕様から，あえてもう一度言いますが費用のかかるインプリメンテーションの前に特性を計算できなかった場合には，そのようなプロジェクトはほとんどまたは全く信頼することができません。この方面には，取り組むべき課題がたくさんあるように思います。もっとも，取り組むべき研究も当然たくさんあります。

プログラミングと同時に，しばしば同一人物，一般的には開発担当者が，私がソフトウェア工

学と呼ぶところの活動を行います。すなわち，計画，監視，制御および品質保証を行います。ソフトウェアが実際に使用されるようになってから，環境が依然として記述された要求項目に準拠しているか監視と制御を行います。ソフトウェア工学では，基本的には，設計ドキュメント，すなわちソフトウェアを物理的なオブジェクトと見なして，それを実行，観察，および合成します。

ここでもう一度，一連の定説が必要になると思います。製品が期待あるいは要求に一致していると言う場合，および各種のバージョンで新しい製品を構成する場合，また他にもいろいろな場合が考えられますが，ソフトウェアは依然として要求と仕様を満たしていると主張します。このような主張を，計算された証拠によって裏付けできない限り，そのような製品はほとんどまたは全く信頼できません。もう一度申し上げますが，現在我々がこの点に関してできることはほとんどありません。

以上のことを述べて最初の話題，つまり一般論の部分，また３つの定説の話を締めくくりたいと思います。機械工学の分野では，費用のかかるインプリメンテーションの前に，たとえば船舶設計に関して計算します。また電子工学技術者は，費用のかかるインプリメンテーションの前に，データ通信設計について計算します。土木技師は，橋梁設計に関して計算します。船舶，人工衛星または橋梁について契約を行うクライアントは，その計算結果を検討して，設計の妥当性を保証のために，保険会社に支援を期待します。

要求項目とソフトウェア工学において，他の分野と比較しても現在ではそのような計算をもっとうまく実行できます。ただ，誰もそんな計算をしているようには見えません。

以上で最初の話題は終わりですが，これはPASCAL派から見た従来のアルゴリズム指向のソフトウェア開発に関するものでした。

さて，アルゴリズムに基づく開発と知識に基づく開発の比較という話題に入りましょう。

まず一般的な例を挙げます。鉄道に関するコンピュータシステムがあると想像してください。(Fig.1) そのシステムでは，列車の運行スケジュールのプランニング，鉄道運行のリアルタイムな管理，切符の発行や予約などのサービス業務などを行うことができます。

そのような鉄道コンピューティングシステムでは，もちろん鉄道に関係なくても構いませんが，コンパイラアプローチを取ることができます。この場合，たとえば新幹線と多くの新幹線停車駅など，特定の鉄道の具体的な内容はコードに隠されます。路線の停車駅の数がある特定の数となるという事実は，同じようなステートメントの繰り返しがその数だけ表れるという形でコードに埋め込まれてしまいます。

インタープリタのアプローチでは，鉄道線路，列車，運行スケジュールなどを固定のデータ構造と考え，計算は解釈に従って進みます。すなわち，インタープリタは汎用的です。異なる固定データ構造を使用することにより，インタープリタを，別の鉄道システムに関して再使用できる場合があります。

２つのアプローチを比較いたしますと (Fig.2)，コンパイラとは，インタプリタを部分評価したものと言うことができます。つまり，インタープリタのコンパイル済みコードは，鉄道システムに関する推論を表現しません。その代わりに，コンパイル済みコードとインタープリタは計算を表現し，鉄道管理においては運動学の数学的法則に従って計算するプログラムになっており，またスケジュールプランニングにおいてはオペレーションズリサーチの法則に従って計算するプログラムになっています。

知識に基づくアプローチでは，これらの法則について考えなければなりません。つまり，プ

鉄道コンピューティングシステム：プランニング、整合管理、サービスおよび管理

3つのアプローチ：

1. コンパイルされたアルゴリズムのアプローチC
鉄道の具体的事項はコード、すなわち、定数やコードの構造および文の合成構造に隠ぺいされる。

2. インタプリトによるアルゴリズムのアプローチI
線路、列車、運行スケジュールなどは固定データ構造で表現され、計算はインタプリトを通して進む。
インタプリタ（I）は汎用である：同じインタプリタで多種多様な鉄道システムを扱うことができる。
実行には時間がかかるが、運行スケジュール、線路、列車などの変更が容易である。

Fig.1

1-2 コンパイルおよびインタプリトされたアルゴリズムA
コンパイラは、インタプリタの部分評価の結果と考えられる。
コンパイルされた個々のインタプリタは、鉄道システムに関する推論を表現できない。
その代わり、計算、運動力学の古典的な数学法則、およびオペレーションズリサーチを表現することができる。

法則は、コンパイル済みコードへ「組み込まれ」、特定のインタプリタへ組み込まれる。

3. 知識に基づくアプローチK
法則を論理形式のルールで表し、鉄道システムの構成要素を事実で表す。
「計算」は、推論マシンMによって進められる。実質的には広範囲にわたるシステムにたいして同じMが使用される。

Fig.2

ランニング，スケジューリング，列車制御の管理を支配するものが何であれ，このような法則はルールとして論理的な形式で表されなければならず，またインタープリタのアプローチで，固定データ構造として与えられたものは，ファクトの集合として考えなければなりません。

さて，ここでは，計算は推論マシンによって進行します。基本的には，その同じ推論マシンは，単に鉄道システムだけではなく，もっと幅広いシステムに使用できます。

基本モデルが意味するものの例としては，アルゴリズム型モデル指向の仕様を挙げます。(Fig.3) そして，それにいくつか付け加えれば，おそらくもう1ページか10〜15行分を追加すれば，皆様が鉄道システムについて知りたいこと，たとえば，特定の時間に発着する列車の運行スケジュール，駅間の異なる長さの線路を示すグラフ，つまり線路システムがあること，

基本モデル領域：A

1.0	*rs:RS*	=	$Time \times SCH \times G \times TS$
2.0	*sc:SCH*	=	$S_m\,(S_m\,(T_m\,(D_m\,A)))$
3.0	*g:G*	=	$S_m\,(S_m\,\mathrm{N}_1^{+})$
4.0	*ts:TS*	=	$\mathrm{T}_m\,(LOC \times KIN)$
5.0	*lo:LOC*	=	$S\,\mid\!(S \times \mathrm{N}_1 \times S)$
6.0	*k:KIN*	=	$L \times V \times AD$
7.0	*ve:V*	=	$\mathrm{N}_0 \times \underline{s}\text{-}max{:}\mathrm{N}_1$
8.0	*ad:AD*	=	$\mathrm{INTG} \times \underline{s}\text{-}max{:}\mathrm{INTG}$
9.0	*Time*	=	$Week \times Day \times Hour \times Min$
10.0	*Week*	=	$1 \mid 2 \mid \ldots \mid 52$
11.0	*Day*	=	$0 \mid 1 \mid \ldots \mid 6$
12.0	*Hour*	=	$0 \mid 2 \mid \ldots \mid 23$
13.0	*Min*	=	$0 \mid 1 \mid \ldots \mid 59$
14.0	*D,A*	=	...
15.0	*s:S*	=	TOKEN
16.0	*t:T*	=	TOKEN
17.0	*le:L*	=	N_1

Fig.3

列車は線路上ないしは駅のどこかに存在していること，列車には一定の長さ，速度，加速度があることなどを知ることができると思います。

まず，このモデルを基に比較してみましょう。(Fig.4) 最初の比較です。モデル指向のアプローチと知識に基づくアプローチでは，同じ問題に対して基本的にどのように異なったアプローチをするのか比較してみようと思います。モデル指向アプローチでは，領域方程式を記述します。ところが，古典的なProlog型では，適切な述語を記述し，また節形式で表現される各種の述語間にある不変式が成り立たなければならない，といったような一貫性制約がなければならないと思います。

この2つのアプローチでは，運行スケジュールの不規則性を異なる方法で取り扱います。(Fig.5) 運行スケジュールが不規則ならば，モデル指向アプローチの場合と同じように，駅と列車の各組み合わせに対して発着時刻を示す節が必要になります。運行スケジュールがかな

表現A	
鉄道運行スケジュールなど	
A	アルゴリズム的開発手法
1	領域方程式： $SCH = S_m\,(S_m\,(T_m\,(D_m\,A)))$ De-Curried: $SCH = (S \times S \times T \times D)_m\,A$

対

表現K	
K	知識工学
1	5項述語として与えられる運行スケジュール： $SCH\,(S, S, T, D, A)$ および指定される引数型： $S \times S \times T \times D \times A \rightarrow$BOOL 到着時刻に対する関数制約条件を 一貫性制約条件としてホーン節を用いて表現可能 $error\,() \leftarrow SCH\,(S_1, S_2, T, D, A')$ $\&\ SCH\,(S_1, S_2, T, D, A'')$ $\&\ A' \neq A''$

Fig.4

所見

運行スケジュールが不規則ならば、*SCH*は事実の節の集合として与えられる：

$$SCH(s_1, s_2, t, d, a)$$

運行スケジュールが「かなり」規則的ならば、*SCH*を、次の節形式ルールとして与えることできる：

18.0 $SCH(S_1, S_2, T, D, A) \leftarrow$
.1 $TRAINS(S_1, S_2, T, FST, ITV, LST, LAG)$,
.2 $INTERVALS(FST, ITV, LST, D)$,
.3 $A = LAG + D$

*INTERVALS*は、定期的な運行スケジュールに従ってDの可能な値をすべて生成する。
事実の節は、*INTERVALS*術語を展開する部分演繹推論によってこのルール形式から導出できる。

Fig.5

り規則的であれば，毎日，１週７日間，１年365日そのまま運行スケジュールを維持できます。したがって，列車の発車間隔と列車が一定の距離を走行するのに要する時間を示すような適切な述語を考案することができると思います。

一般的な比較に移る前に，最初の比較を行ってみましょう。

できれば客観的に申し上げたいと思いますが，知識に基づく形式は，たとえば事実をどのように調べるかという点に関しては，中立的かつ中間的であるような気がします。探索述語の５つの引数は同じ種類です。６つの引数を考えることができ，また入出力は対称的です。しかし，実際にはそんなことはありません。アルゴリズム形式で，領域をモデル化する際に私が選択したデータ構造は，最も一般的な使われ方に適合したものになっているはずです。

駅と列車，および発車時刻を指定されて到着時刻を計算すること以外の標準的でないアクセス形式は，アルゴリズムアプローチではむしろ長々とした指定を必要とし，しかも，それは理解しにくいものです。ただし，アルゴリズムアプローチでは，効率的に詳述化することができます。

スケジューリングについてですが，列車を乗り換えて旅行したいといろいろ考えている乗客ならば，アルゴリズムアプローチよりも，Prolog言語での知識に基づくアプローチの方がより洗練された述語を指定できると思います。また再スケジューリングは，スケジューラの規則に対する何らかのメタ推論の形式によって達成できるでしょう。

この点を一般化し，２つのアプローチを比較してみたいと思います。この比較は，話の始め

に用いた鉄道システムの例について行うのではありません。

次のスライドをご覧下さい。(Table 1) アルゴリズムアプローチでは，入出関数を指定しますが，その関数は後で数学的な意味で実現されます。したがって，インプリメンテーションの関係，詳細化の関係などが問題となります。モデル指向アプローチ，すなわちアルゴリズムアプローチでは,SETS,直積,トリプレット,マップなどのデータ集合構造体を使用します。また，アルゴリズムアプローチにおいては，仕様言語と具体的なプログラミング言語とは明確に区別します。

知識に基づくアプローチでは，記述と仕様を１つずつアサーションとして公式化します。計算の結果は，論理的結論に対応する証明であると考えます。また，仕様言語は，同時にプログラミング言語であるという二元的な見方をします。

我々の作業の根底には，いくつかの仮定があります。(Table 2) アルゴリズムアプローチでは，プログラムは仕様とは同じではないという区別を行います。そして，仕様は実行可能プログラムへと段階的に詳細化されること，それは１人１人のプログラマとソフトウェアエンジニアの仕事であること，および仕様は必ずしも実行可能でなければならないということはないというふうに，はっきりと区別しています。

データ構造仕様のインプリメンテーションに重点が置かれており，最後にコンパイルを行うという重要な側面があります。

知識に基づくアプローチでは，論理はオブ

Table 1

表1A	
形式的なソフトウェア仕様のアスペクト：目的	
A	アルゴリズム的開発手法
1	インプリメントされるべき入出力機能の数学的な方法による仕様定義
2	集合、直積、タプル、マップなどのデータ構造を使用
3	抽象的仕様と具体的プログラミング言語との区別

対

表1K	
K	知識工学
1	アサーションによる現実世界の関係の記述と仕様定義
2	Kの論理的結果に対応する証明として理解される計算結果
3	論理を仕様とプログラミング言語としてとらえる二重の見方

Table 2

表2A	
形式的な仕様のアスペクト／仮定	
A	アルゴリズム的開発手法
1	プログラム≠仕様
2	仕様は、段階的に実行可能プログラムへ詳細化される。仕様は、必ずしも実行可能ではない
3	スタック、待ち行列などのデータ構造の仕様のインプリメンテーションを重視
4	最後にコンパイル

対

表2K	
K	知識工学
1	対象言語としての論理 プログラムと仕様はしばしば混同され、また部分的に同一視される
2	実行可能仕様の仕様＝宣言的プログラミング （参照）Kowalski：アルゴリズム＝論理＋制御
3	領域知識を重視
4	最後に、制約条件の充足および知的なバックトラッキングなどの一般的な手法による効率の達成

ジェクト言語であり，プログラムと仕様はお互いに混同されたり部分的に同一視されることがあります。一方，アルゴリズムアプローチでは，仕様は詳細化されます。Robert Kowalskiの言葉を借りれば，実行可能仕様は，アルゴリズムの意味では論理と制御を合わせたものに等しいと言えます。

次のポイントですが，アルゴリズムアスペクトのリストの中では，基本的にはソフトウェア開発に焦点を絞っています。確かに，私は最初にアルゴリズムアプローチにおいて，要求項目の作成には山積みの課題があると主張しました。要求項目の作成では，領域の知識に焦点が当てられていました。知識に基づくアプローチでは，領域知識に焦点を当てつづけます。しかし，アルゴリズムアプローチによるソフトウェア開発では，より効率的なインプリメンテーションに焦点が移行します。この点については，後でまた触れます。

ここで，コンパイルは，知識工学アプローチにおける制約条件の充足と知的バックトラッキングに相当します。

この表は，計算のある数学的形式に焦点を当てています。(Table 3) アルゴリズムアプローチでは，計算の概念は，関数の概念，命令型プログラミング言語の命令文による段階的な詳細

Table 3

表3A	
計算されたオブジェクトの数学的形式	
A	アルゴリズム的開発手法
1	計算の関数的な概念 段階的に詳細化され、最終的には命令型プログラミング言語の命令文およびデータオブジェクトに対する演算から成るアルゴリズムに落とされる ⇒ 決定性コンパイル
3	計算の基礎としてのλ計算リダクション 仕様はしばしば「関数的」（代数的）である。

対

表3K	
K	知識工学
1	計算の関係概念 ⇒ 関係データベースとのリンク 非決定的（「バックトラッキング」） （擬似）並列性 ⇒ ルール演繹推論（融合） 計算の基礎としてのユニフィケーション
2	アサーションからの答の導出（|=、|-） ⇒ 演繹推論 アブダクション（因果分析） 帰納推論（機械学習）
3	仕様はしばしば、（M:N）関係を強調する「関係」構造として与えられる。

化の概念に相当します。これは，ある意味では決定論的なコンパイルの概念に導かれる概念です。その背後には，何が起こっているかを理解するための計算体系(calculus)の概念があります。

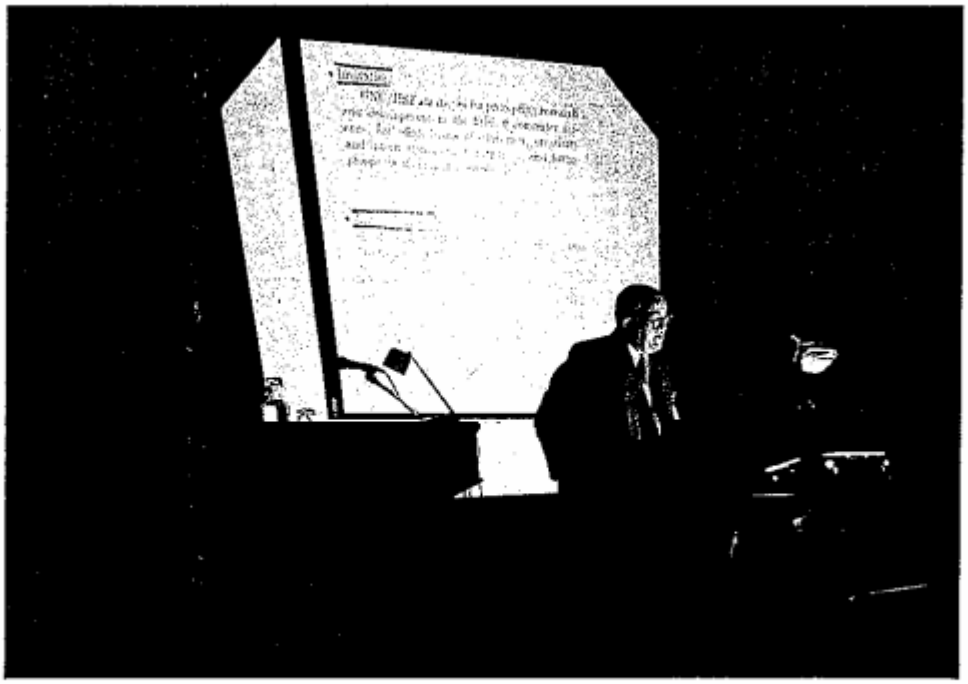
それに対して，知識工学アプローチ，すなわち知識に基づくアプローチでは，計算について，関数的な概念よりも関係的な概念を持っています。これは，私たちが関係データベースへのリンクを見る理由になります。すなわち，関係のアスペクトから，非決定論的な点で考えるのが至極当然になります。その結果，ユニフィケーションが計算の基礎となります。

さらに，演繹推論，アブダクション，帰納推論という，もっと興味深くかつ多種多様な計算オブジェクトがあります。

アルゴリズムアプローチでは，仕様を開発した人は問題領域を理解している人であるのが普通であり，しかも効率的なインプリメンテーションを作成するように依頼を受けるのもこの人です。

コンパイル自体からはもはや効率の向上を望めない場合には，アルゴリズムアプローチで開発を行っている開発担当者は段階的な開発手法で効率性を確保するという責任を負っています。(Table 4) 普通は，非形式的で非自動的な方法で，仕様からプログラムを導き出します。再帰的なループから，反復ループを開発するという方法があります。パラダイム的な例としては，静的意味と動的意味の形式仕様からのコンパイラを開発するという概念があります。

これとは対照的に，知識に基づく仕様は，理想的には実行可能仕様であり，部分リダクション，制約条件の充足，そして特にユニフィケーションを特色とする，巧妙に考案された推論マシンを通して効率性を追求します。



Dines Bjørner氏

Table 4

表4A	
計算の効率化	
A	アルゴリズム的開発手法
1	段階的開発（詳細化） 仕様→...→プログラム 非形式的または（半）自動的 データの抽象化と適用可能形式を重視、（一例として）集合に関する演算はリストに対する演算になる。
2	再帰形式からの反復形式の開発
3 4	パラダイムシステム、例： 静的意味および動的意味の形式的仕様からコンパイラを開発

対

表4K	
K	知識工学
1	仕様は、理想的には実行可能。 たとえば部分演繹推論、制約条件を充足する手法、および特殊なユニフィケーション手法による効率化
2	ルールをトップダウン的（後向き）に使用するか(再帰形式を参照)、またはボトムアップ型（前向き）に使用するか（反復形式を参照）に関しては、最初は一方に決められていない。 メタインタプリト方式の選択によってどちらかが選ばれる。
3 4	パラダイムシステム、例： 演繹推論データベース＝真の宣言性および停止性（決定可能性）が達成可能なProlog論理のサブセット

論理プログラムは，最初は，ルールをトップダウン型とボトムアップ型のどちらを使用するかに関してどちらも選択されていません。アルゴリズムアプローチに関するパラダイム的システムにはコンパイラの例がありましたが，ここでは演繹推論データベースがその例です。

型の概念においては，この２つの型概念は互いに干渉しています。(Table 5) １つは，標準MLに見られるように，パラドックスを回避することに関係してきます。この型の概念は，基本的にはBertrand Russel型の理論に基づいています。知識に基づく工学における型概念は，

Table 5

表５：　驚くべきことに、再帰的に定義、合成、または構成される型を提供する論理プログラミング体系はほとんど存在しない。

表5A	
（データ）型	
A	アルゴリズム的開発手法
1	保護および表現の選択としての伝統的な型（例：整数および浮動小数点）
2	構造化メカニズムとしての型： 多様型 抽象データ型

対

表5*K*	
K	知識工学
1	「現実の世界」のエンティティの分類としての型 ⇒ 階層構造の一般化としての用語論理（概念論理）、三段論法形式、順序ソート論理

よりアリストテレス的であり，エンティティの分類と見なされます。当然のことながら，この２つはオーバラップしていますが，その背景は異なります。

アルゴリズムアプローチでは，保護，コンパイル時の情報，表現の選択として型を使用します。一方，知識工学アプローチでは，各種の論理，用語論理，三段論法形式などあらゆる種類の論理を，分類階層の一般化として扱います。

知識に基づくアプローチでルールと事実が非決定であるのに対応して，アルゴリズム的な開発では，仕様のユニットは，指定された関数に対して非決定的であり得ます。また，アルゴリズムアプローチでの合成原理は，関数合成ですが，それに対して知識アプローチでのそれは，節の連言です。

このことから，この２つのアプローチを比較するもう１つの方法，すなわち各種の構造的メカニズムを比較する方法が導き出されます。(Table 6) まず，要求項目のモデル化，すなわち資源，機能，動作，安全性，性能などに対する，私のきめの細かい段階的アプローチを思い出してください。このような要求項目は何らかの構造を得るのに役立ちましたが，現在の

我々の理解ではそれを数学的に説明することはできないと思います。

知識アプローチでは，データベースの世界のスキーマや継承分類階層などを考えることができます。アルゴリズムアプローチでは，ブロック構造体，モジュール，および手続きという意味での構造化メカニズムの例が数多くあります。一方，知識工学アプローチでは，利用可能な構造化メカニズムははるかに少ないようです。

例外は重要な役割を持っています。鉄道システムの例に戻りますが，列車の運行スケジュールは何らかの規則性を想定しています。したがって，その場合には例外はありませんでした。運行スケジュールを繰り返す間隔が与えられていない場合には，運行スケジュールをすべての例外で展開することによって，運行スケジュールをアルゴリズムアプローチで修正しなければならないことがあります。または，データ構造を詳述することによって，間隔を拡大するだけで修正することもできます。

知識に基づくアプローチでは，ルールの爆発的な増加を避けるために，規則的な運行スケジュールを例外で上書きすることによって，非単調論理の各種の性質を利用することができる場合があります。(Table 7) アルゴリズム的な開発では，明示的な列挙に基づいて例外を取り扱います。一方，知識に基づくアプローチでは，当然のことながら，「Aではない」ということを導出できるということと，Aを導出できないということとは基本的に異なります。

私のプレゼンテーションの第2部の終わりにやって参りました。第1部で扱った例，すなわ

Table 6

表6A	
記述的な構造化メカニズム	
A	アルゴリズム的開発手法
1	粒度の細かい要求項目モデル 基本、機能、動作；環境およびインタフェース、安全性の臨界値、依存性、CHI
2	ブロック構造およびカプセル化機構付きモジュール手続き

対

表6K	
K	知識工学
1	データベースの概念モデルとスキーマ継承分類階層構造 仕様の「自己充足」単位としてのルール節
2	ホーン節

ち鉄道システムの例に戻ります。ルールと事実に関して推論する推論マシンについて述べました。固定データ構造で機能するインタープリタについて述べました。さらに同じシステムに関して，コンパイル可能なプログラムについても述べました。基本的には，ルールと事実を推論マシンに適用し，インタープリタと固定データ構造の組み合わせを基本的に導出する，ある種の部分評価器を期待できるかもしれません。

同様に，インタープリタと一定のデータ構造に適用可能で，かつ私のプログラムを生み出してくれるような部分評価器を考えることができます。(Fig.6) そのような部分評価器は，一定の法則を満足しなければなりません。その法則とは，以下の通りです。

ルールと事実に適用される推論マシンの意味は，固定データ構造に適用されるインタープリタの意味と同じでなければならず，しかも私の

Table 7

- または、元のスケジュールを「修正」、すなわち常に一年間にわたるスケジュールを最初から作成する：

$$23.0 \quad SCH = Time_m\,(S_m\,(S_m\,(T_m\,(D_m\,A))))$$

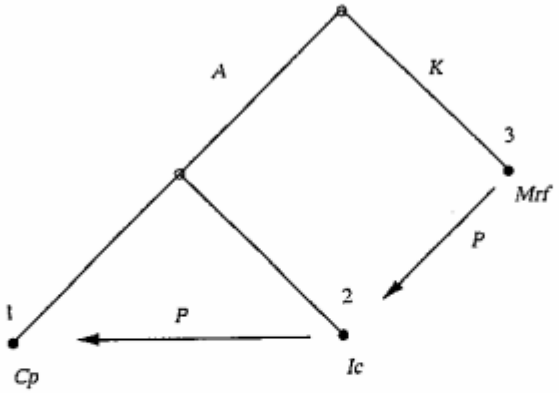
- 非単調論理： 例外で「上書き」することによって定期的なスケジュールを修正する。
- これによって、例外が定期的なスキーマに入り込むのを回避する。

表7A	
領域モデルの例外と非単調性	
A	アルゴリズム的開発手法
2	事例の直接的な列挙によって取り扱われる例外

対

表7K	
K	知識工学
1	「（有限）失敗としての否定」（SLDNF導出）
2	⊢¬ Aと⊢/Aとの違い

C-I-Kアプローチ



P： 部分評価器
機能は以下の法則を満足しなければならない：

- $[M]_L rf = [I]_L c = [p]_L = [[C]_L' p]_M$
- $([P]_L Mr.f) \fallingdotseq (I.c)$
- $[P]_L Id \fallingdotseq p$

凡例：

- p: （コンパイル対象の）鉄道システムプログラム
- c: 鉄道システムの固定データ構造（インタプリト対象）
- f: 鉄道システムの構成要素を反映する知識に基づく事実
- r: 鉄道システムのルールを反映する知識に基づくルール
- C: コンパイラ
- I: インタプリタ
- M: 推論マシン

二村の写像

[[mix P S] D] = [P] S D
T = [mix] I P
C = [mix] MIX I
CG = [mix] mix mix

Fig.6

プログラムにも同じ意味を与えてくれるものでなければなりません。もちろん，推論マシン，インタープリタ，およびプログラムを，同じL言語で作成することができます。これは，私のプログラムに適用可能な他の言語で書けるコンパイラの意味と同じでなければならず，さらに何らかの機械語で書かれたプログラムの意味と同じでなければなりません。もちろん，これは満足しなければならないアスペクトの1つです。次に，部分評価器が満足しなければならないルールがあります。最初の部分は，基本的なインタープリタを与えてくれるはずの推論マシンとルールに適用される部分評価器を意味しています。ある種の同形表現単位は，固定データ構造で，というよりこの場合は鉄道システムに関する事実の間で満足されなければなりません。それがすべて同じ言語で書かれているならば，等号になります。同様に，部分評価器は，インタープリタに適用される部分評価器の意味を適用しましたが，これは同じものを与えてくれなければなりません。

さて，これは非常にすばらしいことですし，非常によいように思えます。しかし，問題はその部分評価器，Pをどのように構成するかということです。これは，もちろん，即答するのが難しい問題です。もしその答を知っていたら，おそらく私はこの場に立っていないと思います。ですから，その答を見つけ出すのを助けてもらうために，今皆さんにお話ししているわけです。そのような部分評価器を構築するための経験が蓄積されています。部分評価器は，各種の法則を満足しなければなりません。

その中で最も古典的な部分評価器が，この一番下に示してあります。MIXは部分評価器です。そして，その概念は，プログラムとその静的データの意味に適用されるMIX,動的データに適用されるMIXの意味，すなわちその意味は他の言語で書かれたプログラムの意味に等しくなければならず，おそらく静的データと動的データに適用されるということです。

部分評価器の意味をMIXに適用するインタープリタとソースプログラムが与えられると，ターゲットプログラムを獲得できます。それが与えられると，ルール，つまりMIXが満足しなければならない法則を書くことができます。MIXとインタープリタに適用されるMIXの意味によってコンパイラが得られ，MIXに適用されるMIXの意味とMIXによってコンパイラジェネレータが得られます。さて，今ご紹介している知識は20年前からあるものです。おそらく，これはNeil Jones教授によるとより洗練された記述ができると思いますが，これについてはよく知っておくべきだと思います。それは二村の写像として知られており，この最後の3つの等式がそうです。

問題は，この部分評価器，Pをどのように構成するのか知りたいということです。(私が数多くの表を比較してからこの点にたどり着いたということは，ある意味では，正確さを期すのはもちろんですが，Pが具体化しなければならない，確かにこの最初のPはまさしくメタなものを具体化しなければならないのです。ここで「知識」という用語を使うべきではないのですが，Pは，事実，すなわち我々がアルゴリズムアプローチと知識に基づくアプローチとの関係に関して知っている事柄を具体化しなければなりません。）ですから，各表で各種の比較を行ったことから，部分評価器が何とか表現されています。これが私が申し上げたい点です。

これで私の話の第2部が終わります。それは，最初の部分よりは技術的な話でした。最後に，国連大学について，並びに国連大学がマカオの研究トレーニングセンターを6月8日，来週月曜日からスタートすることに決定したことについて少しお話しします。UNUIISTは，ポルトガルと中国の両国政府並びにマカオ政庁から3

千万ドルの寄付を受けております。この研究トレーニングセンターの活動は，発展途上国に目を向けたものになります。その活動の幅の広さはかなり野心的であると思います。ソフトウェアの使用法，ソフトウェア技術管理，ソフトウェア開発，カリキュラム開発，および研究活動において支援することになります。それが実現できればと願っておりますが，数々のプロジェクト，コース，研究，顧問活動，および普及活動を通して具体的に実現していくことになります。

UNUIISTでは，３種類のプロジェクトを計画しています。１つは，新しいアプローチやアプリケーションの実現可能性を調査するために，研究的なアイデアを小さいながらも難しい部分集合に適用するという予備的な小規模プロジェクトです。このようなプロジェクトは，デモンストレーションのプロジェクトになるかもしれませんし，またそれが別の研究につながる可能性もあります。デモンストレーションのプロジェクトは，我々が伸縮自在と期待している最先端の技術をアプリケーションに適用するとともに，教育コースの基礎にもなります。最終的には，これは，技術移転プロジェクトに発展する可能性があります。つまり，UNUIISTとしては自らソフトウェアは開発しませんが，プロジェクトが最終的には発展途上国で新たに登場，または既存の研究グループに移転されるように現地の人々を動員すべきであると考えております。

トレーニングは，複数のプロジェクトとコースの組み合わせから成り立っています。トレーニング，啓蒙，および教育の３種類のコースを提供します。具体的には，トレーニングコースでは，アプリケーション，ソフトウェアのインストール，操作，および使用法を取り上げます。また，啓蒙コースでは，ソフトウェア技術の管理に重点を置き，教育コースではこの講演でこれまでお話ししてきた内容に重点を置きます。つまり，教育コースは，具体的には，産学界からの大学院レベルの技術者向けの３ヶ月間の専門コースです。

研究も行いますが，それはこの研究トレーニングセンターの主な活動ではありません。先進工業国が生み出してきた成果が発展途上国で実現できるようになればと心から願っておりますし，また反対に，たとえば中国やインドで生まれた多数の大きな期待の持てる成果やアイデアが，先進工業国で幅広く知られるようになるべきであるとも思います。最初は，我々のスタッフや研究奨学生を現地に派遣して，我々自身の研究を行います。ただし，ほとんどの場合，発展途上国の研究機関や産業界と共同活動を行うことになります。まず，ソフトウェア開発のアルゴリズムアプローチに焦点を当てるとともに，基本的にはduration calculusとして知られる連続時間インタバル時相論理に焦点を当てます。

大学のグループ，ソフトウェア企業および他の研究センターを１つに結ぶために，何らかの有機的なネットワークが構築されるでしょう。

マカオ研究トレーニングセンターでのプロジェクト，並びに通常のコース，短期コース，セミナー，長期滞在型集中コースに参加する産学界の大学院レベルの技術者に対して，奨学金を提供します。また，マカオで研究活動を行う学生や研究者に対しては，６ヶ月から９ヶ月にわたる奨学金制度を設けています。

基本的には，UNUIISTには多くの特色があります。国連体制に関しては，顧問活動を行うことができればと思っています。実際に，いくつかのワークショップやパネルを今秋開催することが，すでに計画に入っています。研究活動を行い，また各種のコースを設け，さまざまなプロジェクトにも取り組みます。このようなことを通して，発展途上国の研究機関の各種の部門と関係を持つことになるでしょう。

私の話も，締めるときがやって参りました。

UNUIISTの研究開発の推進戦略の一部を述べましたので，これで私が言うべきことの義務は果たしたのではないかと思います。アルゴリズム的開発並びにアルゴリズムアプローチと，知識に基づくアプローチの部分では，我々の研究開発の背後に存在する姿勢をご紹介しました。この会議で講演をするようにご招待を受けたとき，創造性と将来の展望について話してほしいと言われました。時間の関係で，ここではそのお話はいたしませんでしたが，提出した論文に出ております。

ここにいらっしゃる司会者の田中英彦教授と，私を招待してくださった古川康一教授に感謝の意を表明したいと思います。また，今回は，準備の面で神林さんに大変お世話になりました。また，論文を共同で執筆してくれた同僚にも感謝します。それでは，皆様，ご静聴ありがとうございました。

［質疑応答］

Wolfgang Bibel：Bjørner教授，アルゴリズムと知識の部分の比較に関して，私は教授とは非常に異なった意見を持っていることを認めなければなりません。基本的には，私の最初の質問は，知識工学の部分で，教授は非常に視野の狭い見方を取っているのではないかという点です。見たところ，教授は知識工学を，単にPrologプログラミングとして考えていらっしゃると思います。それは，私には非常に視野の狭い見方に感じられます。なぜならば，プログラムの抽出，つまり証明から実際のプログラムを抽出することを考えてみてください。または，与えられた問題の知識仕様と，抽象データ型を統合化すること考えてみてください。そのようなことは，まだ他にも数多くあります。効率的な現実のプログラムへのコンパイルでさえ，多数の研究者がプログラム合成において目標としていることです。あるいは，これらのことはあまり広く知れわたっていないのかもしれません。しかし，それを認識すべきだと思います。教授がこの２つのもの，つまり単にアルゴリズムを一方に置き，知識工学を他方に置くのであれば，それは歪んだ見方であると思います。これが私の最初の質問です。視野が狭すぎると思いませんか。

もう１つの質問は，知識工学は幅広い視点を持っており，その視点の１つはプログラムの修正を行うことができるということです。すなわち，知識仕様をただ修正すれば，その変更は，合成されたプログラムに反映されます。私の質問は，アルゴリズム部分には何かそれに匹敵するものはないのかということです。また，アルゴリズムアプローチを使用するプログラマは，すべてのことを最初からやり直さなければならないというのは本当なのでしょうか。

Bjørner：分かりました。あなたのご質問は２つではなく，少なくとも３つあると思います。まず，あなたの最初の質問の前半部分をお聞きしたとき，ほとんどのことに「はい」とお答えするつもりでした。確かに，私は知識工学に対して視野の狭い見方をしております。すなわち，数学的に，あるいは閉じた形式で何もかも正確に書き表すことができるという見方です。しかし，あなたの最初の質問の後半部分に対しては，「いいえ」とお答えしなければなりません。証明からのプログラムの抽出に関して指摘された面は，実際に私の比較に含まれていると思います。同じことが，帰納推論的プログラミング，アブダクティブプログラミングでも言えると思います。ここで重要なのは，私がコンパイラやデータベースシステムの開発を依頼した開発担当者は，すべての効率性をインプリメントする責任を負っているということです。一方，問題領域の簡潔な論理形式をあなたが行うような方法で考え出す問題領域モデル作成者は，使用の

効率性に関する責任を負っていません。そのような責任を求めるのは，融合法やユニフィケーションに関して巧妙なアルゴリズムを考案したり，また証明などからプログラムを抽出するために数多くのことを考え出す他の人々です。

あなたの２番目の質問では，あなたの意見に歴史的な観点からのみ賛成したいと思います。非常に正確な銀行業務の要求項目モデルを開発できる場合で，そしてあらゆる新しいソフトウェアと既存のソフトウェアの改訂がまさにそのモデルに従っている場合には，どのような変更も非常に簡単に導入できるような同形態が得られます。ただし，アルゴリズム化しなければならない点は変わりません。とはいえ，元に戻って古い内容を変更する必要はありません。これは，より技術的な話です。

E.A. Feigenbaum：国際ソフトウェア技術研究所がマカオのようなきわめて風変わりではあるものの魅力的な場所に設立されることになった，歴史的な経緯を教えていただけますか。

Bjørner：マカオには資金があったからです。1985年に委員会が発足し，その委員会の派遣メンバーがシンガポール，香港，マカオ，韓国の４カ所を訪問しました。そしてマカオの人々，マカオ政庁は即座に資金提供の約束を申し出てくれたのです。香港の方がマカオよりも魅力があると考えられるかも知れませんが，マカオには，香港には見られない，すばらしく落ち着いたポルトガル的な特徴があると考えていいと思います。

基本的には，そのような研究機関を設立するニーズがあることを確認してから，資金の提供元を求めて国々を歩き回った結果，マカオ政庁が３千万米ドルの資金を提供してくれることになったのです。この点については，次のような状況を申し上げるべきでしょう。つまり，国連大学は，他にも数々の研究機関を所有しております。フィンランドのヘルシンキに世界経済の発展を研究する研究所が１つ，オランダのマーストリヒトに新技術，特に新技術の社会的な重要性について研究するための研究所が１つ，他にも発展途上国を重視する研究所もあります。これらの研究所は，すべて発展途上国に関係する諸問題の解決の責任を負っております。それらの所在地はすべて先進工業国であるにもかかわらず，その役目は発展途上国の諸問題を扱うことだったのです。ですから，新しい研究所は発展途上国に設立すべきものであると考えたのです。そしてそれが実現されたのです。

個人的にはマカオは魅力的な場所であると思います。とはいえ，歓楽街やギャンブルなどのことを言っているのではありません。マカオはすばらしい場所です。ぜひ，マカオの研究所においでください。