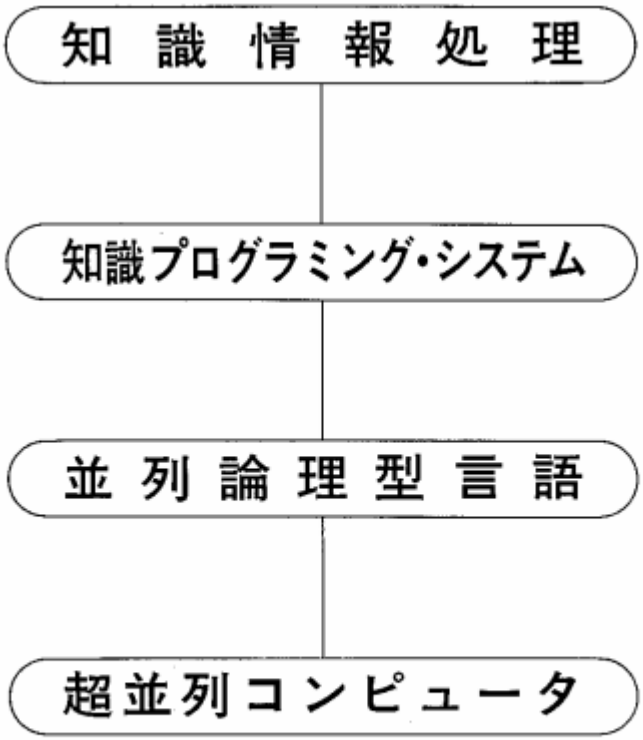
基礎ソフトウェア・システム

ICOT研究所 第1研究室
室長 古 川 康 一



基礎ソフトウェア・システムの主なテーマと成果

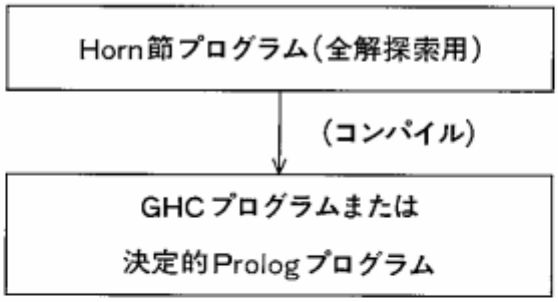
1．5G核言語の設計・開発

- 並列論理型言語GHC
- 言語階層KL 1

2．問題解決・知識表現

- メタ・プログラミング
- 知識獲得と仮説推論
- 対話型問題解決

GHCによる全解探索処理



意　義

(1) 並列探索に一般に必要な多重環境を，コンパイルによって消去する。
このときextralogicalな機能を用いない。

(2) 問題によっては，多重環境を用いた探索より効率が向上する。
リストの分解で30倍，順列生成で6倍（Prolog処理系での比較）

(3) 探索の制御が可能になる。

GHCプログラムの
アルゴリズミック・デバッギング

プログラムの動作とは独立な抽象的性質だけを用いる。

バグの存在する範囲を機械的に絞り込む。

- GHCプログラムの抽象的意味の導入
- 対処可能なエラー
  - 間違った答を返して停止する
  - デッドロック
- Divide & Query戦略

  Query総数 ～ $O(\log_2 N)$　　N：手続き呼出し総数
- GHCで実現されている。

KL1-b

設計思想

(1) 当面Multi-PSIを強く意識

・処理単位をKL1のゴール・レベルに設定

・PE内の逐次性を利用した最適化

(2) 実用性

・デバッグ機能

・高い信頼性

(3) システム・プログラムの記述を考慮

・ハードウェア操作組込述語

PSI上の逐次処理系

1．目　的

・Multi-PSIにおける機械語(KL1-b)の設計及び部分試作

・モジュール化機能(KL1-u)の検討及び試作

・デバッグ機能の検討及び試作

2．現　状

・KL1-u/cのKL1-bへのコンパイラ及びKL1-bのエミュレータをESPで記述 → 約0.9kRPS

・Multi-PSI用の処理系に拡張中

汎用計算機上の逐次処理系

目　的

(1) 汎用機上に真に実用的な処理系を実現する。

(2) それによってGHCの普及をはかる。

(3) 逐次計算機むきのコンパイル・最適化技法を研究・開発する。それを並列計算機上の実現にも役立てる。

現　況

・目的コードの設計，性能予測

予測性能：　"append"：　$32\times\alpha$ kRPS

クイックソート：　$19\times\alpha$ kRPS

素数生成：　$15\times\alpha$ kRPS

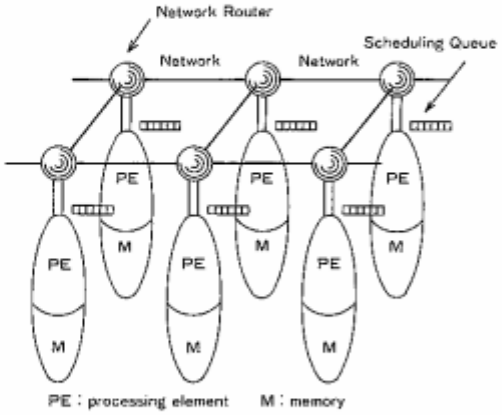
(α：　閉じたサブルーチン化による速度低下)

分散処理系ソフトウェア・シミュレータ

目的：① 分散処理系の基本動作の確認

② プログラマによる実行制御の動作確認

③ 統計情報(処理ゴール数，処理サイクル等)の収集

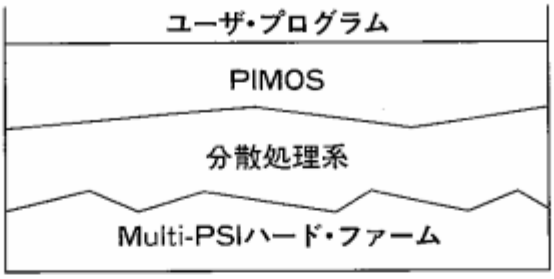


★それぞれのPEは独立したメモリを持っている。

★それぞれのPEはScheduling Queueを持っている。

PIMOS

★PIM/Multi-PSI用のオペレーティング・システム



★機　能

① 実行時の負荷の調整(ロード・バランス)

② 資源，メモリの管理

③ オブジェクト・コードの分配・管理

④ ユーザ・タスクの管理，OSのシェル

⑤ 入出力，割り込み，割り出しの管理

GHCの応用プログラム

Propositional Temporal Logicの定理証明

Omega-Graph Refutation

temporal formulaのvalidityを判定する手続き

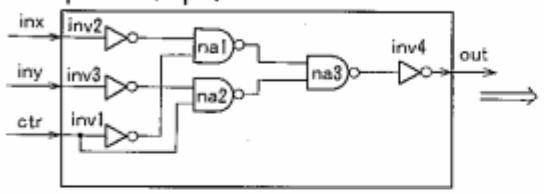
≡ Omega-Graph中でOmega-loopを探すこと

GHCによる実現

- Omega-Graphを互いに通信し合うプロセスのネットワークとして実現
- Omega-loopの検出をプロセス間のメッセージの送受により実現
- Omega-Graphの生成とOmega-loopの検出をオーバラップさせることにより高い並列度を抽出している。

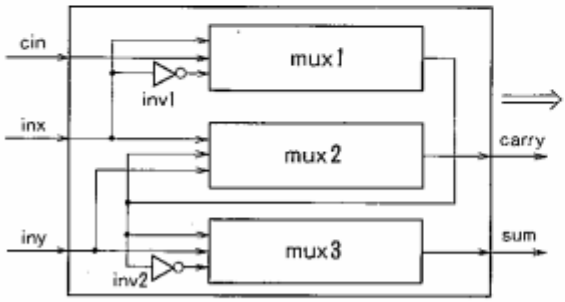
GHCによる論理シミュレータの試作

multiplexer(mpx)




```
mpx(Name, T, Inx, Iny, Ctr) :-
      true|
    nand_2in(Name-na1, T, M2, M3, M4),
    nand_2in(Name-na2, T, Ctr, M1, M5),
    nand_2in(Name-na3, T, M4, M5, M6),
    inv(Name-inv1, T, Ctr, M2),
    inv(Name-inv2, T, Inx, M3),
    inv(Name-inv3, T, Iny, M1),
    inv(Name-inv4, T, M6, Out).
```

full_adder(fadd)




```
fadd(Name, T, Inx, Iny, Cin, Carry, Sum) :-
      true|
      inv(Name-inv1, T, Inx, M1),
      inv(Name-inv2, T, M2, M3),
      mpx(Name-mpx1, T, inx, M1, Cin, M2),
      mpx(Name-mpx2, T, Inx, Iny, M2, Carry),
      mpx(Name-mpx3, T, M2, M3, Iny, Sum).
```

並列構文解析

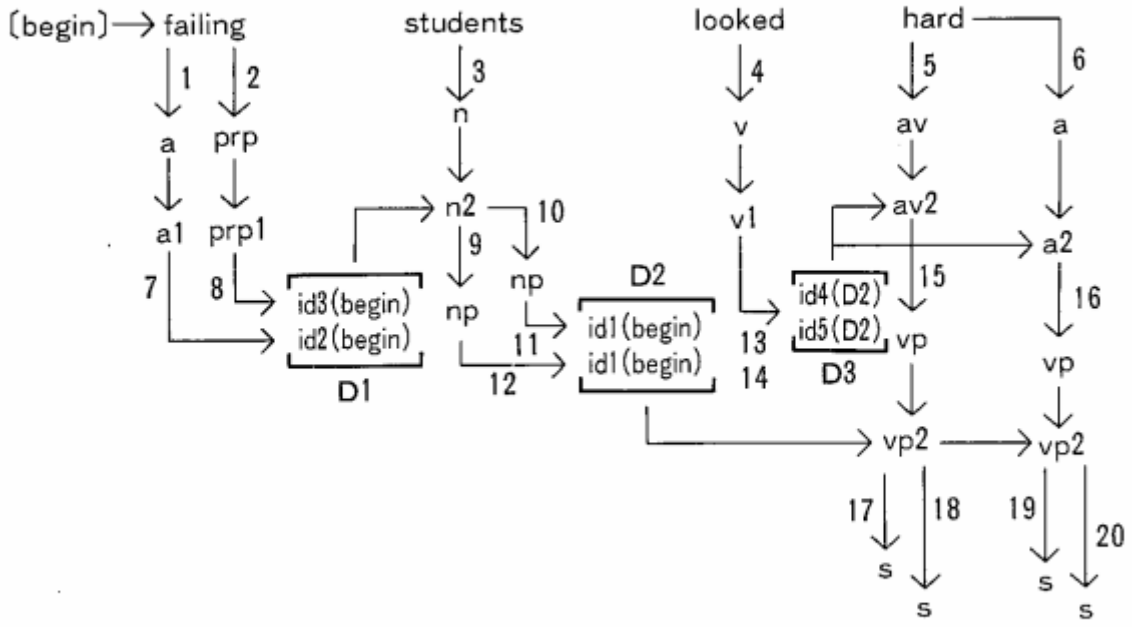
自然言語文法（文脈自由形式） → トランスレータ → GHCによる構文解析プログラム

- 単語，品詞などの構文要素：プロセスとして定義
- プロセス間通信による新しいプロセス(構文要素)の生成
- 基本戦略：

  ボトムアップ(小さな要素をもとにより大きな要素を作り上げる)

  ＋トップダウン予測(プロセス間を渡るデータのフイルタリング)
- BUPと比較して5～10倍速い

並列構文解析



failing(〔begin〕, D1), students(D1, D2), looked(D2, D3), hard(D3, D4), fin(D4).

メタ・プログラミング

推論についての高階の記述を可能にする。

問題対象に独立な記述を可能にする。

推論システムを構築する上で重要な概念を自然に表現できる。

ex. 推論のメタ・レベルからの制御

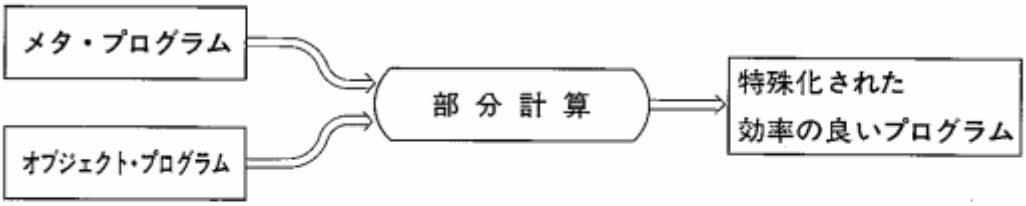
推論過程の観察

推論過程の説明

問題点

実行効率の低さ

部分計算によるメタ・プログラムの高速化



意義：メタ・プログラミングを実用的なものにする方法を示唆

効果：
- メタ・レベル，オブジェクト・レベルの分離によるプログラムの理解し易さ，変更のし易さ等の実現
- 高い実行効率の実現

メタ・インタプリタの段階的特殊化

オブジェクト・プログラムが段階的に構成される場合（例，エキスパート・システムのルール・ベース）に，メタ・プログラムを段階的に特殊化する方法

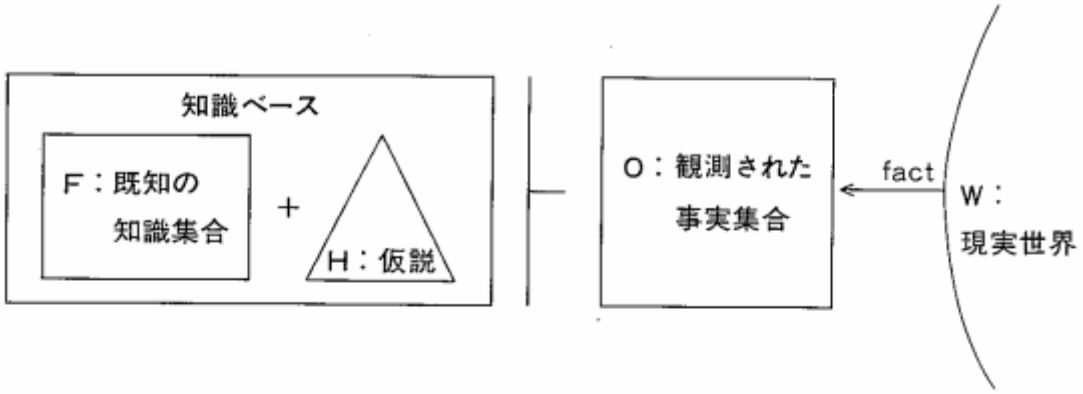
段階的に構成されるオブジェクト・プログラム $P=\{P_1, \cdots, P_m\}$

$\{P_1\} \rightarrow (P_1, P_2) \rightarrow \{P_1, P_2, P_3\} \rightarrow \cdots \rightarrow \{P_1, \cdots, P_m\}$

メタ・インタプリタ I の段階的特殊化

Partially Specialized Compiler：Com_I^k

仮説推論の枠組

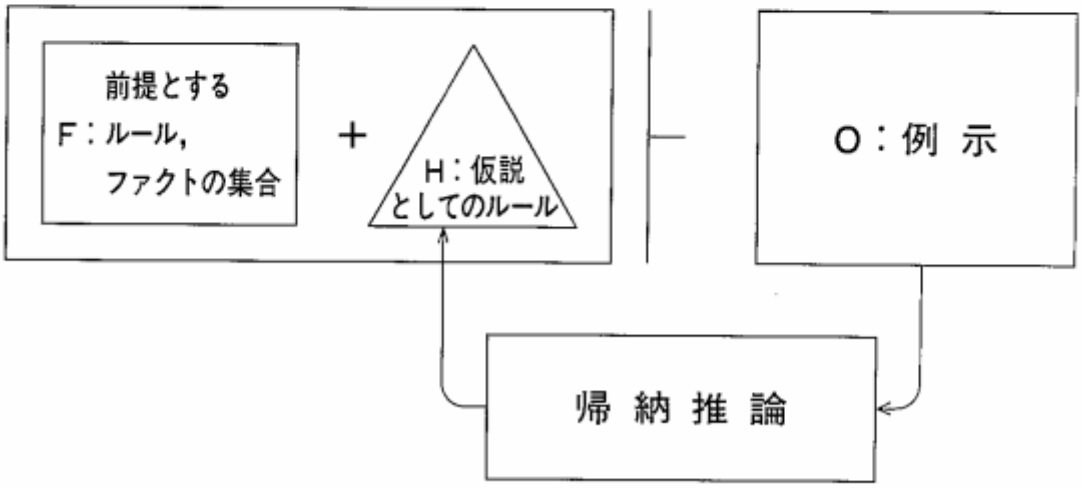


1．Fは矛盾していない。

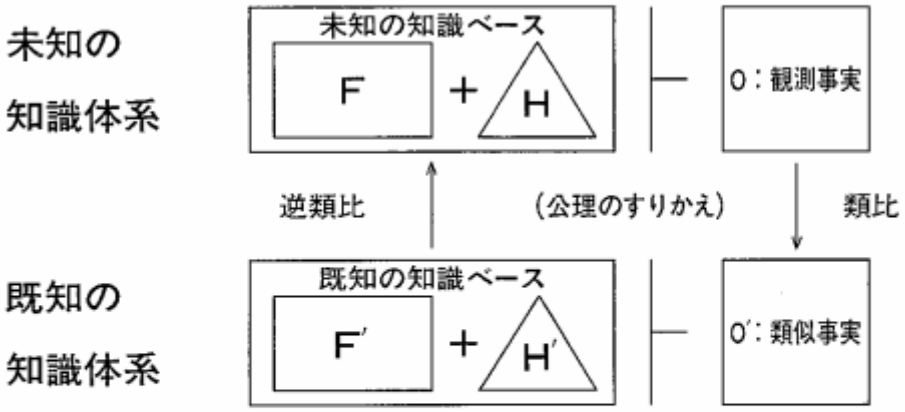
2．Fだけからoを説明できない。

3．Fに矛盾しないある仮説Hを考えると，
FとHからOが説明できる。

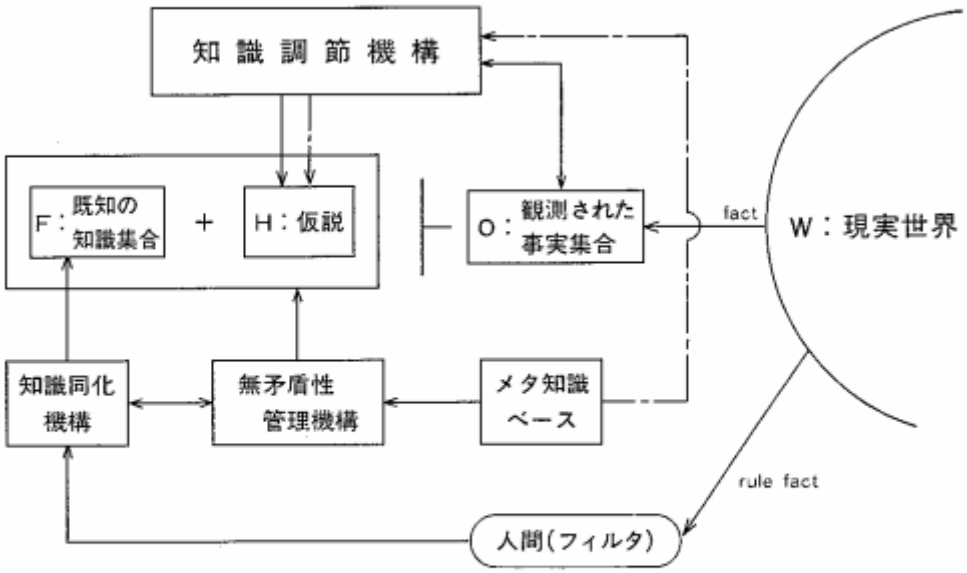
仮説推論と帰納



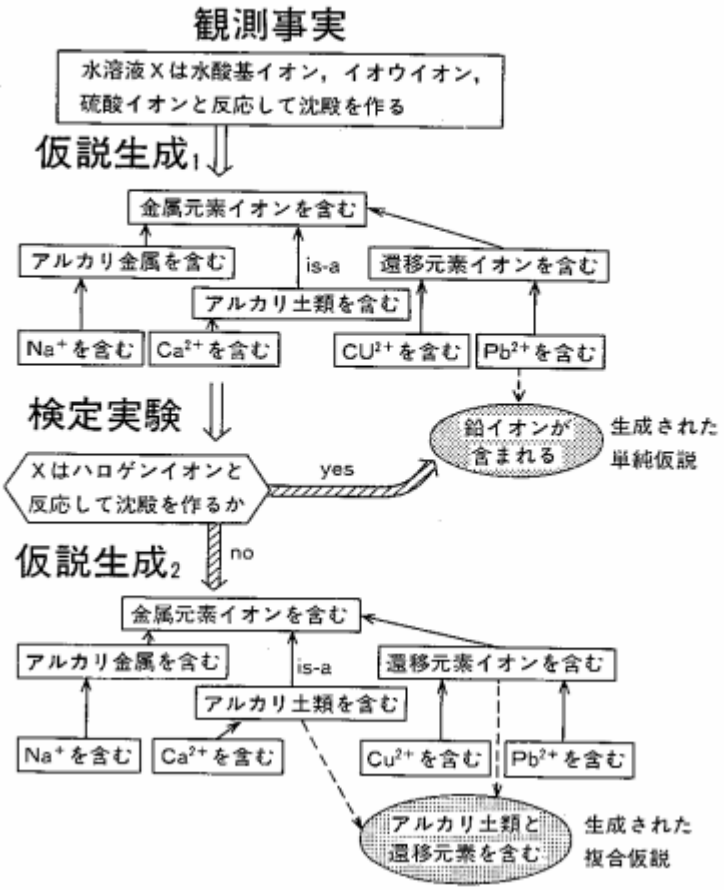
仮説推論とアナロジー



仮説推論と知識獲得



仮説生成・検定実験の例



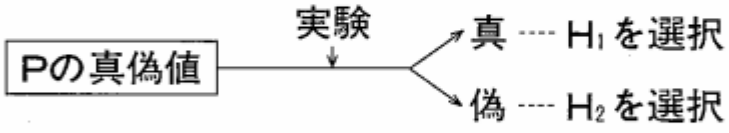
仮説選択のためのテスト生成法

◦ 仮説H_1，H_2が区別可能とは

「$F+H_1 \vdash P$ かつ

$F+H_2 \vdash \neg P$」という命題(観測事実)Pが存在する。

◦ 仮説選択



◦ このようなPをどうやって見出すか？

⇩

- 仮説選択の問題をホーン論理の枠組で定式化
- Shapiroのdivide-and-queryアルゴリズムの変形により，効率的に命題を発見

インタラクティブなシステム

(1) ユーザとの頻繁で柔軟なインタラクション

⇒ Interactive Query Revision

(2) 双方向の宣言的関係

⇒ コンストレイント・プログラミング

(3) ユーザ・フレンドリ・インタフェース

対象分野に大きく依存

Interactive Query Revisionの実現

(1) queryのインタラクティブな追加・変更

⇒ query stackを使ってqueryの保持・管理

(2) query間の論理的整合

⇒ query stackを使ったバックトラックのシミュレーション

コンストレイント・プログラムの実現

(1) 双方向の宣言的関係の記述

⇒ CILのfreeze機能を使った数値計算のコンストレイントの記述

LSI配置問題への応用