TM-1202

Quixote システム

安川　秀樹、津田　宏、横田　一正、田中　学、
森田　幸伯、平井　千秋、合田　光宏、
高瀬　真司、高橋　千恵、小鳥　量、
西岡　利博、鈴木　隆一、鎌田　進一

August, 1992

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

Quixote システム

安川 秀樹　津田 宏　横田 一正　田中 勉　森田 幸伯　平井 千秋
ICOT 第3研究室　沖電気工業(株)　(株)日立製作所

合田 光宏　高瀬 真志　高橋 千恵　小島 量
(株)アーティフィッシャル・インテリジェンス　(財)JIPDEC　(株)管理工学研究所

西岡 利博　鈴木 隆一　鎌田 進一
(株)三菱総合研究所　三菱電機東部コンピュータシステム(株)

概要

知識ベース / 知識表現言語 *Quixote* は、知識情報処理システム開発に必要とされる知識ベース機能、知識表現機能を提供する中核的な言語を目標として設計・開発されている。

Quixote システム第1版は、*Quixote* の基本言語仕様の検証と、実装アルゴリズムの確認を目的としたプロトタイプであった。その評価に基づき、言語仕様の改良、および応用システムでの実際の利用を目的とした第2版処理系の設計・開発を行なった。本稿は、その報告である。

言語仕様に関しては、第1版の評価および法的推論・自然言語処理を主な対象とした検討結果から、特に大きな改良・拡張の必要はないとの結論に達した。ただし、リスト・集合については、応用システムでの必要性を考慮し、新たに追加する方針をとった。また、言語の意味論については、前年度に与えた宣言的な意味論に加え、処理系の設計・開発と連動し、手続き的な意味論を検討することとした。

第2版処理系の設計・開発に関しては、

- 応用で必要となる各種インタフェース機能、提供機能の拡充
- 永続データ管理、トランザクション、更新など、第1版処理系で除外した機能の実装
- KL1/PIMOS 上での効率的実装のためのシステム設計、アルゴリズム設計

を中心に、研究開発を推進することとした。

1　はじめに

知識ベース / 知識表現言語 *Quixote* は、知識情報処理システム開発に必要とされる知識ベース機能、知識表現機能を提供する中核的な言語を目標として設計・開発されている。

Quixote は、知識情報処理システムの開発において、2つの側面を持つ。1つは、知識ベース管理基本プログラム Kappa の上位層に位置づけられ、Kappa の提供する非正規関係に基づく拡張され

たデータベース機能を基に、知識ベースを構築するための機能を提供する知識ベース言語としての側面であり、もう1つは、知識ベースと応用プログラムのインタフェースを確立し、応用プログラムに知識表現・操作機能、推論機能を提供する知識表現言語としての側面である。

知識情報処理システム開発においては、主な要求事項としては、

- 知識表現モデルの拡張： 複雑かつ複合的な情報・データ、部分的にのみ定義されているような情報・データを効率的に表現することができる汎用的な表現体系
- 推論モデル： 上記のような情報・データの操作、あるいは、それらに関する柔軟・高度な推論を可能とする計算モデル
- 知識の統合化：複数の個別に定義されたデータベース・知識ベースを融合・統合するための体系

が挙げられる。これらの要求事項に対応するため、*QUIXOTE* では、

- 知識表現モデル： 基本的な知識オブジェクトを表現する基本オブジェクト、知識オブジェクトの属性を構成要素とする、再帰的な複合項表現に基づくオブジェクト指向的な知識表現体系の導入、
- 計算モデル： 論理型言語、演繹データベースなどの演繹指向の体系に、複合項によるデータモデルの拡張、複合項間の汎化関係(包摂関係)に関する制約領域の導入を図ったオブジェクト指向＋制約論理型という計算パラダイムの採用、
- 知識の統合化： 知識オブジェクトをモジュールの識別子とするモジュール概念、モジュール間の階層関係、モジュールの結合演算による、知識のモジュール化と階層化、ならびに、知識の継承概念の導入

を図った。*QUIXOTE* は、そのオブジェクト指向性、モジュール概念によって、応用システム開発の際に発生する、問題領域と知識ベース / 知識表現との間の意味的ギャップを低減することが可能であると同時に、制約論理型パラダイムに基づく推論機能によって、高度な演繹機能を提供することができる。

2 システム第2版の特徴

本年度は、上記の開発方針に基づき、*QUIXOTE* の言語仕様の見直しと第2版処理系の設計・開発を行なった。

言語仕様に関しては、第1版の実験結果から、基本的な部分は法的推論、あるいは、自然言語処理という応用に対して十分であると結論した。したがって、*QUIXOTE* の基本的な言語仕様とその宣言的な意味は、前年度と同じである [成果報告 91-1]。

ただし、第1版処理系では未対応であったリストや集合などのデータ構造の扱いを中心に、言語の手続き的な意味が不明確であった部分の見直しと改良を行なった。リストに関しては、特殊なオブジェクト表現に対する簡略表記として扱うこととし、従来の論理型言語と同様なリスト操作と併せて、リ

スト間の包摂関係に関する扱いも可能とした。また、ここでいう集合は、本来 *Quixote* で扱っている Hoare 順序による選言的な (disjunctive) 集合ではなく、連言的 (conjunctive) な集合である。ただし、出現しうる場所を限定しているため、リストと同様に構文的な要素 (簡略表記) として扱われるため、言語仕様上は大きな修正を要さないものである。

一方、処理系設計・開発に関しては、基本的には第 1 版から完全に書き直しを行なった。その主な理由は、

- 第 1 版は主要機能、基本アルゴリズムの確認を主目的としたラピッドプロトタイピングであること、
- 第 1 版の評価の結果、効率的実装のためには、システム構成、データ設計レベルからの改良が必要であることが判明したこと、

による。また、応用システムの開発に際して、ユーザインタフェース機能、プログラムインタフェース機能の拡張が必須であることが明かとなったため、処理系の提供機能の拡充を行なった。具体的には、

- 端末インタフェースの拡張 (QMACS エディタを介した対話機能)
- プログラムインタフェースの改良・拡張 (データ形式、呼び出し形式の整備)
- 知識オブジェクトの束構造、モジュールの階層構造に関する情報提供機能の拡充
- 質問に対する解の導出過程の表示機能の追加

などを行なった。

また、第 1 版では、外部ファイルや外部データベースとのインタフェース機能が未着手であったが、第 2 版では、

- 永続性 (persistence)
- トランザクション、更新処理

を扱うことができる。

以上に挙げた処理系の拡張により、*Quixote* の言語機能を応用プログラムで簡便に利用できるとともに、Kappa の提供するデータベース機能を *Quixote* で利用することが可能となった。

第 2 版処理系の実装は、第 1 版と同様、KL1/PIMOS 上に行なったが、第 1 版の開発経験をもとに、並列化に適したアルゴリズムの検討を行ない、設計・開発を進めた。実装に関しての課題は、

- 単一化操作・制約解消の動的性質
- 実行時の環境としての多重環境の必要性
- 属性継承に関連して発生する多大な処理

など、KL1/PIMOS 上での実装を考えた場合、本質的に効率化が難しい側面にどう対処するかということである。これらの問題に関しては、データ設計や実装アルゴリズム、あるいは、モジュール構成などの実装技術レベルで対処し、効率化を図った。

知識ベース / 知識表現言語に関する研究開発は、今後の知識処理システム開発のための中核的な技術となるものと考えられる。そういった観点から考えると、今後の課題として、以下のようなことが挙げられる:

- より大規模な応用事例での有効性の検証
- 並列アルゴリズムの改良などの効率化
- *Quixote* の効率的な実装手法
- プラットホーム機能、プログラミング機能などに関する *Quixote* の言語仕様の拡張
- 応用システムの開発環境とのインタフェースの確立

また、実用的な言語処理系としては、実行効率の向上は必須であり、上記の並列アルゴリズムの改良などとともに、KL1/PIMOS 上での効率的な実装を前提とした言語仕様の変更・修正という方向を検討する必要があるものと考えられる。

以下、3 章では、*Quixote* 第 2 版処理系の全体構成、4 章から 7 章では、それぞれインタフェース、オブジェクト・マネージャ、インタプリタ、制約コンパレータという各モジュールについて、その概要ならびに仕様を説明する。また、*Quixote* 第 2 版処理系の提供する機能ならびに *Quixote* の構文的仕様を付録に示した。

3　システム第2版の全体構成

3.1　概要

Quixote の全体のモジュール構成の概要は以下のようになっている。

- インタフェース (IF)
- オブジェクト・マネジャ (OM)
- インタプリタ (IP)
- 制約コンパレータ (CC)

この中で IF は、ユーザ、永続記憶、実行系との3種類のインタフェースと、それらのコーディネータとからなっており、*Quixote* の環境の設定を行う。それらは以下のモジュールから構成されている。

- 通信マネジャ (CM; Communication Manager)
- 永続記憶マネジャ (PM; Persistent Memory Manager)
- スーパバイザ (SV; Supervisor)
- 多重通信マネジャ (MCM; Multiple Communication Manager)
- サーバ (QS; Quixote Server)

OM は *Quixote* が使用するデータの管理を一括して行う。IP は *Quixote* のインタプリタで、導出木の生成と解のまとめあげを行う。CC は制約に関するサブルーチンの集合で、制約の簡約化、比較、単一化などを行う。

3.2 でこれらの関係を説明する。

3.2　モジュール構成図

DB を一つしか使用しないときと、複数の DB を同時に使用するときとで構成が異なっている。

① 単一 DB しか使用しない場合: この中で、QS, PM, Kappa (太字) はシステムに唯一つしか存在せず、複数ユーザから共有されるが、DB (CM, SV, OM, IP, CC) はデータベースごとに生成され、各ユーザに割り当てられる。ただしいったん生成されたデータベースは QS によって再利用される。

② 複数 DB を使用する場合: この中で MCM が各 DB へのディスパッチを行う。各 DB (CM 以下) は単一 DB 使用と同じである。

3.3　データの流れ

Quixote で使用するデータには何種類かあり、図-3.3-1のように変換されている。これらの中で、KL1 プログラムから出すことのできるメッセージについては、7.4 節の 7.4 で、テキスト形式の構文については、7.4 節の 7.4 で説明する。

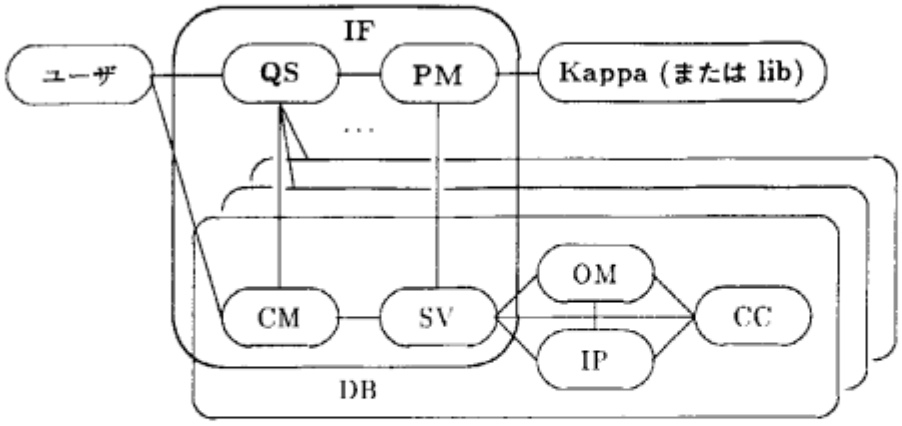


図 3.2-1 単一データベースを使用するときのモジュール構成図

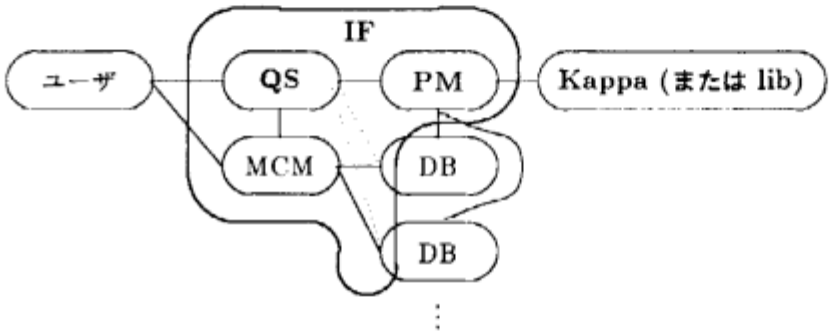


図 3.2-2 複数データベースを使用するときのモジュール構成図

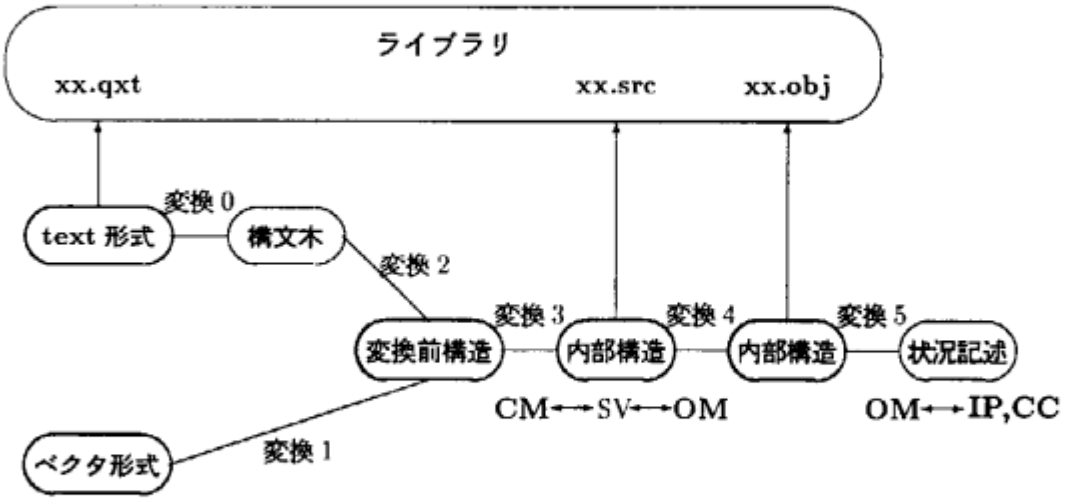


図 3.3-1 データの流れ

4　インタフェース (IF)

4.1　概要

(a) ユーザ に対して、*Quixote* を利用するための端末インターフェースを提供する。

(b) ユーザ に対して、プログラム から、*Quixote* を利用するためのプログラム インターフェースを提供する。

(c) サーバ機能をもち DB およびユーザ管理をおこなう。

(d) 各提供機能において、処理系内部で利用できるように、データ変換をおこなう。

4.2　モジュール構成図 (概要)

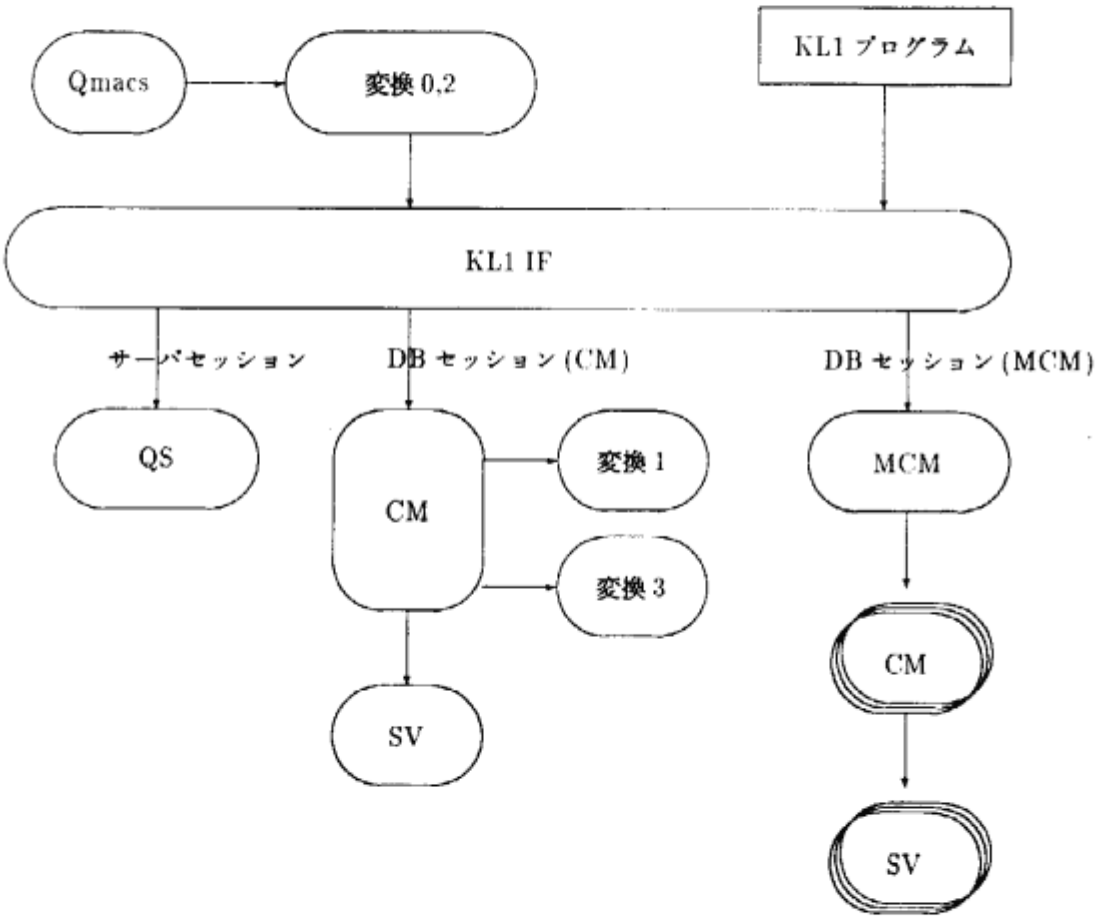


図 4.2–1 インタフェース (IF) 部 モジュール構成

4.3 各モジュールの機能概要

① Qmacs

- Pmacs をベースにした エディタ型ユーザインターフェースであり、*Quixote* の機能を Qmacs のコマンドとして提供する。

② KL1 インターフェース

- Kl1 プログラムにたいして、*Quixote* の機能を提供し、サーバセッションでは、QS へ、DB セッション（CM）では、CM へ、DB セッション（MCM）では、MCM へ、メッセージをながすことで処理を実行する。

③ QS

- *Quixote* のサーバ機能を持ち、DB 即ち CM の管理およびユーザ管理 をおこなう。

④ CM

- 1 つの *Quixote* DB の機能を提供し、各機能で、必要に応じて、変換 1 、変換 3 の機能によりデータ変換を実行して、SV にメッセージを流す。

⑤ MCM

- 複数の CM を保持し、各メーッセージにおいて、指定の CM をとりだしてメッセージをながす。

⑥ SV

- OM,IP,CC の生成をおこない、CM からのメーッセージをうけて OM、IP への コマンド制御をおこなう。
- トランザクション ID を生成し、トランザクション管理機能をもつ。
- *Quixote* DB を DBsrc として保持する。

⑦ 変換 0,1,2 変換 0,1,2 は入力を変換前構造に変換する処理である。プログラム、問合せなどが入力となる。変換 0,2 は ESP インタフェースであり、変換 1 は KL1 インタフェースである。

(a) 変換前構造

プログラム、問合せなどを一つのベクタで表現される構文木に展開したものである。ベクタの要素はアトム、ベクタ、ベクタのリストであり、各ベクタは第一要素をタグとしている。プログラムや問合せは以下のように展開される。

- program(env-sect(...), obj-sect(...), mod-sect(...), rule-sect(...))
- query(no-update, proc-mode(...), ans-mode(...), inheritance(...), program(...))

(b) 変換 0,2

変換 0 はパーザ (left scanning parser) を呼びだし、テキスト形式の入力を構文木に変換する。変換 2 は構文木を変換前構造に変換する。構文木は変換 1,2 の内部データであり他のモジュールより見ることはできない。また、変換 0,2 は別別に呼び出すことはできず、変換 0,2 の順に必ず動作するので実質としては一つのモジュールである。変換 0 は、以下の 5 種類のパーザと、それらを呼び出す部分より構成される。

- プログラムの環境 section 用パーザ
- プログラムのモジュール間関係 section 用パーザ
- プログラムのオブジェクト項間関係 section 用パーザ
- プログラムのルール section 用パーザ
- 問合せ用パーザ

1 つのパーザはトークン解析、構文解析、構文木作成の 3 つの部分よりなる。変換 0,2 には以下の述語が用意されている。

- :get(#mae_kozo,program, String,^Vector,^Status)
  テキスト形式のプログラムを変換前構造に変換する。
- :get(#mae_kozo,query, String,^Vector,^Status)
  テキスト形式の問合せを変換前構造に変換する。
- :get(#mae_kozo,m_id, String,^Vector,^Status)
  テキスト形式のモジュール ID を変換前構造に変換する。
- :get(#mae_kozo,b_obj, String,^Vector,^Status)
  テキスト形式の基本オブジェクトを変換前構造に変換する。

(c) 変換 1

変換 1 はベクタ形式の入力を変換前構造に変換する。変換 1 の逆に変換前構造よりベクタ形式のデータに変換する機能が逆変換 1 である。変換 0,2、逆変換 1 を組み合わせてテキスト形式の入力をベクタ形式に変換することができる。変換 1 には以下の述語が用意されている。

- qxtIFcnv1:program(ProgramVector, ^HenkanmaeProgramVector,^Status)
  ベクタ形式のプログラムを変換前構造に変換する。
- qxtIFcnv1:query(QueryVector, ^HenkanmaeQueryVector,^Status)
  ベクタ形式の問合せを変換前構造に変換する。
- qxtIFcnv1:m_id(MidVector, ^HenkanmaeMidVector,^Status)
  ベクタ形式のモジュール ID を変換前構造に変換する。
- qxtIFcnv1:basic_obj(Bobj, ^HenkanmaeBobjVector,^Status)
  ベクタ形式の基本オブジェクトを変換前構造に変換する。

逆変換 1 には以下の述語が用意されている。

- qxtIFrev1:program(HenkanmaeProgramVector, ^ProgramVector,^Status)
  変換前構造のプログラムをベクタ形式に変換する。
- qxtIFrev1:query(HenkanmaeQueryVector, ^QueryVector,^Status)
  変換前構造の問合せをベクタ形式に変換する。
- qxtIFrev1:m_id(HenkanmaeMidVector, ^MidVector,^Status)
  変換前構造のモジュール ID をベクタ形式に変換する。
- qxtIFrev1:basic_obj(HenkanmaeBobjVector, ^Bobj,^Status)
  変換前構造の基本オブジェクトをベクタ形式に変換する。

⑧ 変換 3 変換 3 のサブ・モジュール構成は、図 4.3-1の通りである。

1. 処理

   変換前構造を *QUIXOTE* の内部データ構造のリストに変換する。
   基本方針は、1 つのベクターである変換前構造を上層のベクターから解析し、必要なサブ・モジュールに投げて処理させることである。この時、henkan3_common のメソッドを用い、必要とあれば、変換前構造の変換を行う。

2. 提供メッセージ

   - create_henkan3(Obj,^Henkan3In,^Status)

Obj	::	.src 又は、void
		.src の場合は、そこで用いられている id を内部状態として用い、
		与えられた変換前構造を変換する。
Henkan3In	::	変換 3 プロセスのストリーム
Status	::	プロセス生成ステータス

   - henkan3(Data,^NewData,^Status)

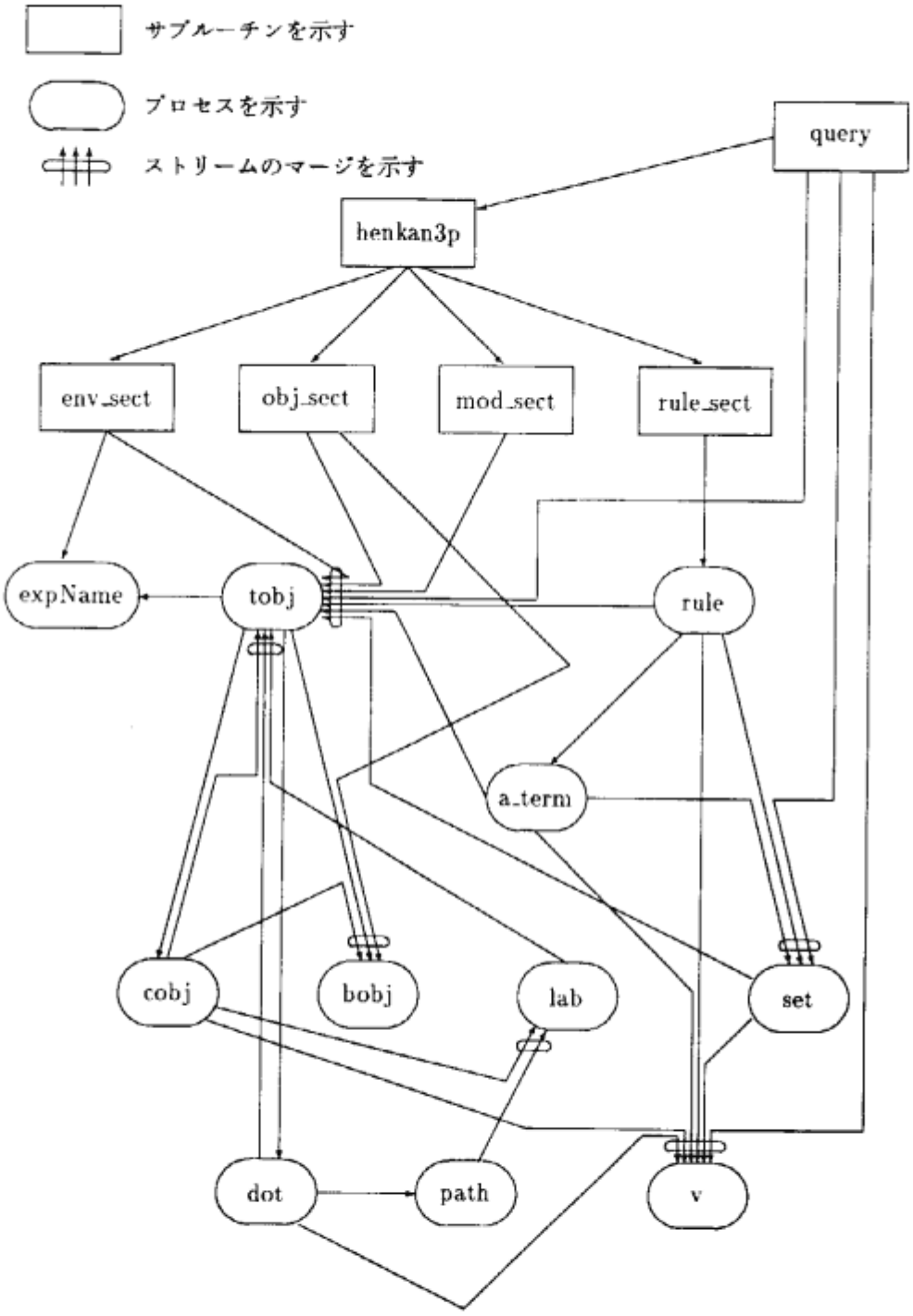


図 4.3-1 変換 3 のサブ・モジュール構成図

Data :: 変換前構造 program

NewData :: program を変換した時に出来た新しい内部データ構造のリスト

Status :: 変換ステータス

- henkan3(Data,^NewData,^Status)

Data :: 変換前構造 query

NewData :: {[Query],DB}

Query :: query の内部データ構造

DB :: query を変換した時に出来た新しい内部データ構造のリスト

Status :: 変換ステータス

- henkan3(Data,^ID,^NewData,^Status)

Data :: 以下の変換前構造

m_id

basic_obj

rule

lab

ID Data に振られた ID

NewData :: Data を変換する時に出来た新しい内部データ構造のリスト

Status :: 変換ステータス

5 オブジェクト・マネジャ (OM)

5.1 概要

本節では、OM 部の概要について述べる。OM では、*QUIXOTE* 言語によって記述されたオブジェクトに関する情報を内部形式で保持し、インタプリタによる推論に必要な諸情報を提供する。

OM では、オブジェクトのファクトおよびルールによる定義、オブジェクトの汎化関係、モジュールの継承関係およびモジュール間の関係などが管理される。また、*QUIXOTE* 第2版では、オブジェクトの実装上の識別のために内部識別子を割り当てており、その管理も OM 内部で行なわれている。さらに、変数情報や、環境情報 (SD) などの CC, IP に対して、推論に必要な内部データ管理機能も提供する。

5.2 モジュール構成図 (概要)

OM では、上記の機能を実現するために、管理する情報の種類などから、以下の9つのサブモジュールからなる構成になっている。

(i) ディスパッチャサブモジュール (DP)

(ii) プログラム管理サブモジュール (PRM)

(iii) 束管理サブモジュール (LM)

(iv) オブジェクト管理サブモジュール (OTM)

(v) モジュール順序管理サブモジュール (MM)

(vi) ルール管理サブモジュール (RM)

(vii) 変数管理サブモジュール (VM)

(viii) ID 管理サブモジュール (IDM)

(ix) SD 管理サブモジュール (SD)

DP を除く各サブモジュールは、DP により生成されその時、入力用のストリームを一つ持つ。各サブモジュールは、入力ストリームからプロトコルに従ったコマンドに対して順に要求に答える。また、各サブモジュールの保持しているデータのロードおよびセーブを行なえるようにする。

各サブモジュールでの共通プロトコルとしては、以下のものがある。

- create(FromStream, ^ToStream, ^Status)
  サブモジュール（マネージャ）の生成
- kill(^Status)
  サブモジュールの終了要求

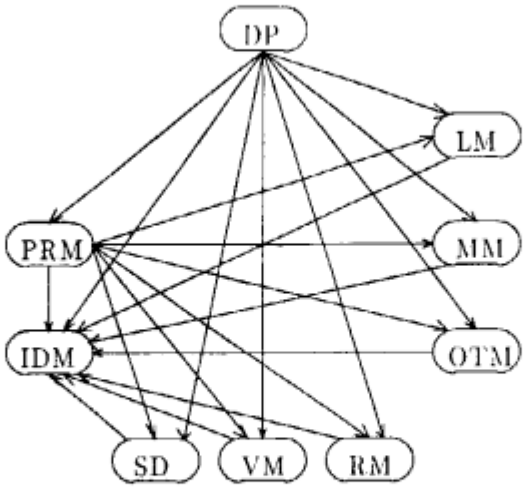


図 5.2-1 OM モジュール構成図

- load_data(Data, ^ID, ^Status)
  サブモジュールのデータの初期化
- save_date(ID,^Data, ^Status)
  サブモジュールのデータの保存

5.3 各モジュールの機能概要

① DP

OM は、*Quixote* の諸情報を各情報の種類により、専用の管理サブモジュールを用いて管理を行なっている。従って、各サブモジュールの初期化や終了処理および OM に対する要求の割当などを行なうサブモジュールが必要となる。これらの処理は、DP サブモジュールで行なわれる。

即ち DP は、

(i) OM の初期化時の、各サブモジュールの初期化および、通信用ストリームの設定、

(ii) OM への要求(メッセージ)の 各サブモジュールへのディスパッチ、

(iii) OM の終了時要求に対し、過去の要求の処理状態を確認の後、各サブモジュールを終了させる、

といった機能を持つ。

上記機能の実現のため、DP では、外部メッセージをサブモジュールへのメッセージに変換するためのテーブル、および、外部メッセージのステータスを保持するキューを利用している。

② PRM

PRM は、OM に送られるプログラムを内部のデータ形式に変換するサブモジュールである。PRM の機能は以下である。

(i) SV からのプログラム／問い合わせを入力として、OM の内部形式へ変換する。

(ii) 変換されたプログラムや問い合わせから関連した情報を、LM, OTM, MM, RM へ送る。

(iii) 問い合わせの解（および各種エラー）を逆変換を行なって、SV に返す。

(iv) SV からの要求に応じて、プログラムを返す。

PRM の主なプロトコルとしては以下がある。

- translate_program_4(TID,Program,^Status)
  プログラム (Program) を内部形式に変換して、必要なサブプロセスに分配する。
- translate_query_4(TID,Query,^DPIDs,^Status)
  問い合わせ (Query) を内部形式に変換する。

③ LM

Quixote では、制約解消時などの基本演算の一つとして、束演算を行なう必要がある。LM では、そのために、基本オブジェクトに対する利用者の汎化の順序の指定から束構造を構成し、CC からの束演算要求に答えるサブモジュールである。LM は、以下の機能を持つ。

(i) SV からの送られてくる基本オブジェクトとその順序から束を構成する。その際新たに作られた基本オブジェクトを PRM に返す。

(ii) CC からの要求に従い、束演算を行なう。

(iii) SV からの要求 (show_Lattice) に従い、束構造を返す。

(iv) SV からの要求 (query, modify_lattice) に従い、束構造を変更する。その際新たに作られた基本オブジェクトを PRM に返す。

上記の機能を実現するために、LM では、基本オブジェクトの順序を表わす行列を保持している。

LM の主なプロトコルとしては以下がある。

- bsubsume(TID, Blarge, Bsmall, ^Res, ^Status)
  基本オブジェクト Bsmall, Blarge の順序を調べる。
- bcompare(TID, B1, ?OP, B2, ^Res, ^Status)
  基本オブジェクト B1, B2 の順序を調べる。

- bmeet(TID, B1, B2, ^Res, ^Status)
  基本オブジェクト B1 と B2 の meet をとる。
- bjoin(TID, B1, B2, ^Res, ^Status)
  基本オブジェクト B1 と B2 の join をとる。
- bcong(TID, B1, B2, ^Res, ^Status)
  基本オブジェクト B1 と B2 が同値かどうかを調べる。

④ OTM

OTM は、以下の機能を持つ。

(i) ある条件を満たすオブジェクト項の検索、

(ii) オブジェクト項の構造的情報の検索、

(iii) オブジェクト項のコピー (変数の名前換え)、

上記の機能の実現のため、OTM では、オブジェクト項の内部識別子、頭部 (head)、ラベルなどをキーとするオブジェクト項テーブルを保持している。

OTM の主なプロトコルとしては以下がある。

- rename(OID, ^NewOID, ^BT, ^Status)
  内部識別子 (OID) で表されるオブジェクト項に現われる変数に新規の変数を割当直し、新しいオブジェクト項を作り、その内部識別子 (NewOID) を返す。
- otm_find_data(Head,Arity,Labels,^IDlist,^Status)
  頭部の基本オブジェクト (Head)、ラベルの数 (Arity)、ラベルのリスト (Labels) を受け、それに該当するオブジェクト項の内部識別子のリスト (IDlist) を返す。
- otm_get_data(ID,^Data,^Status)
  オブジェクト項の内部識別子 (ID) を受け、該当するオブジェクト項に対するテーブルエントリーを返す。

⑤ MM

Quixote のプログラムは、モジュール毎に記述することができ、モジュール間の継承の指定（順序）や特定のモジュール間にまたがる特定のオブジェクトの関係などを定義できる。これらのモジュール間の関係の情報は、MM で管理されている。

MM では、モジュールに関する以下の機能を持つ。

(i) モジュールの順序情報を管理する。

(ii) モジュール間関係の情報を管理する。

MM は、主なプロトコルは以下である。

- show_module(^Mod_Hierarchy_stream, ^Status)
  submodel 関係をストリームで返す。
- create_module(Mod_Id, WhereIs, Rule_stream, ^Status)
  モジュール順序関係の指定された位置 (WhreIs) に、指定された規則 (Rule_stream) からなるモジュール (Mod_Id) を作成する。
- recreate_module(Mod_Id, AbsDel_stream, ^NewMod_Id, ^Status)
  指定されたモジュール (Mod_Id) に 指定された操作 (AbsDel_stream) を順次適用し、できるモジュールを (NewMod_Id) として作成する。
- delete_modele(Mod_Id, ^Status)
  指定されたモジュール (Mod_Id) を削除する。
- modify_submodule_link(Link, ^Status)
  submodule 関係に指定された関係 (Link) を追加する。
- modify_module_link(Link_name, Link, ^Status)
  指定されたリンク (Link_name) をリンク (Link) 情報を追加する。
- submodule(Sub, ^Parents, ^Status)
  指定されたモジュール (Sub) の上位のモジュールを返す。

⑥ RM

RM は、*QUIXOTE* のルールおよびファクトを管理する。IP からの要求により必要なルールおよびファクトを推論処理に必要な形で提供する。

RM は、以下の機能を持つ。

(i) 継承モードに従って、ファクトの継承処理を行なう

(ii) MM の情報に基づき、モジュール間継承を行なう。

(iii) 推論に必要なルール／ファクトの検索を検索し、推論処理に必要な形に変換する。

これらの機能を実現するために、RM では、ルールテーブルを管理している。

IP からの Subgoal にマッチするルール／ファクトの検索要求のプロトコルは以下である。

- get_rule(TID, Subgoal, ^Ruleset1, ^Ruleset2, ^Ruleset3, ^Status)
  サブゴール (Subgoal) と単一化可能な、オブジェクトに対する規則を返す。

⑦ VM

VM は、

(i) 変数情報の管理、

(ii) 変数参照情報の管理、

などの機能を提供し、CC, IP の実行環境を提供する。

上記の機能の実現のため、VM では、変数の内部識別子、変数のクラス (オブジェクト項、集合など)、参照先のデータの内部識別子などをキーとする変数テーブルを保持している。

VM のプロトコルとしては、以下がある。

- rename_var(V, ^NewV, ^Status)
  指定されたトップレベルの変数 (V) をリネームした新規の変数 (NewV) を返す。
- ref(ID1,ID2,^Refered,^Referer,^Status)
  2 つの変数もしくはオブジェクト項の内部識別子 (ID1,ID2) を受け、少なくともどちらか一方が変数の場合には、両者の間に参照ポインタを張る。参照先 (Refered)、参照元 (Referer) を返す。
- deref(Referer,^Refered,^Status)
  変数もしくはオブジェクト項の内部識別子 (Referer) を受け、参照先のデータの内部識別子 (Refered) を返す。

⑧ IDM

IDM は、

(i) 全ての内部データのトランザクション単位での保持、

(ii) 内部識別子と内部データの対応の管理、

(iii) 内部データの追加・検索・修正、

(iv) オブジェクト項、変数の追加・修正等の OTM, VM への通知、

などの機能を持ち、CC, IP に対して、内部データ管理機能を提供する。

上記の機能の実現のため、IDM では、データの内部識別子、トランザクション識別子、データの種別 (オブジェクト項、変数など) などをキーとする ID(内部識別子) テーブルを保持している。

IDM への主なプロトコルとしては以下がある。

- id_add(TID, ID, Class, Tag, Content, ^Status)
  データの内部識別子 (ID) とそのクラス (Class) タグ (Tag) 内容 (Content) を受けて、内部識別子を登録する。
- id_find(TID, ID, ^Class, ^Tag, ^Content, ^Status)
  内部識別子 (ID) を受け、その内部識別子で示されるオブジェクトの情報を取り出す。

- id_new_type_id(TID,Head,Label_Values,^Result,^Status)
  頭部の基本オブジェクト(Head)とラベル-値対のリスト(Label_Values)を受け、既登録のオブジェクト項であれば、その内部識別子を、そうでなければ、新規に登録し、その内部識別子を返す。
- id_correct_id(TID,ID,^Corrected,^Result)
  オブジェクト項の内部識別子(ID)を受け、それが他のオブジェクト項と等価であれば、両者を代表する内部識別子(Corrected)を返す。
- id_get_real_id(Oterm, ^ID, ^Status)
  オブジェクト項(Oterm)を受け、その内部識別子(ID)を返す。
- id_get_field(TID, ID, Field, ^Value, ^Status)
  id_mget_field(TID, ID, [(Field, ^Value), ...], ^Status)
  オブジェクト項の内部識別子(ID)を受け、オブジェクトに関するの (Field などで)指定されたフィールドの情報を返す。
- id_put_field(TID, ID, Field, ^OldValue, NewValue, ^Status)
  id_mput_field(TID, ID, [(Field, ^OldValue, NewValue), ...], ^Status)
  オブジェクト項の内部識別子(ID)を受け、オブジェクトに関するの (Field などで)指定されたフィールドの情報を指定した値に設定する。

⑨ SD

SD は、IP や CC で推論の途中で必要となる情報を保持する環境テーブルの管理に用いられる。SD では、環境テーブルの生成や環境テーブルへの情報の付加や参照、環境テーブル間の関係の管理などを行なっている。

これらを実現する、SD に対するプロトコルとして以下のものが用意されている。

- new(ID, VarTableID, ^ModTableId)
- new_sd(TID, ^NewSD, ^Status)
- new_sun_sd(TID, ParentSD, ^DaughterSD, ^Status)
- sub_sd(TID, ParentSD, DaugherSD, ^Status)
- add(SD, MID, OT, ^Status)
- add_constr(SD, MID, Constr, ^Status)
- hold(SD, MID, OT, ^Res, ^Status)
- hold_constr(SD, MID, OT, ^Res, ^Status)
- kill_all_sd(^Status)
- merge_sd(SD1, SD2, BT, ^NewSD, NewBT, ^Status)

- subsume_sd(Sd1, SD2, BT, ^Res, ^Status)
- cong_sd(SD1, SD2, BT, ^Res, ^Status)

6 インタプリタ (IP)

本節では、*Quixote* のインタプリタ・モジュールで実現した機能と、それを実装したアルゴリズムについて簡単に述べる。各サブモジュールの構成を明らかにし、各サブモジュール間での機能分担について簡単に述べる。

6.1 機能の概要

本インタプリタは、OM からの interpret メッセージに対応して、ルールを駆動し、解を求める。実装上の特徴は、以下の点である。

- 全解探索を行う。複数の解がある場合、一つひとつ返すのではなく、一度にすべての解を返す。
- ルールの展開に対応して、導出木と呼ばれる n 進 AND/OR 木を生成する。その各節で部分解を求め、根に向かって部分解をまとめあげることで解を生成する。
- 導出木の生成の際に、ルックアップという技法が用いられている。これは、ある節の部分解を求める際に、そこを根とする部分導出木を展開することなく、導出木の他の部分から計算済みの解を複写してきてすませるものである。これにより、再帰的なルールのうちのあるものについては、無限に展開を続けることなく終了させることができ、また、冗長な再計算の手間を省くことができる。

6.2 モジュール構成

IP は、大きく 4 つの部分から構成されている。

① 初期化部 (IE:Initial Environment processor)

② 展開部 (RP:Resolution Processor)

③ 解併合部 (SP:Solution-merge Processor)

④ データ管理部 (DM:Data Manager)

IE は、プロセスで、外部窓口として SV,OM プロセスからメッセージを受け付ける。DB セッションの開始 / 終了、トランザクションの開始 / 終了 / 中止に伴うトランザクションの状態の変更と保持を行う。問合せによる推論開始指示を受け付けると RP を呼びだし推論を開始する。

RP は、ルールを適用して導出木を作り上げる。このとき、SP に支持を与えて導出木の各ノードに対応したノードプロセスを生成する。

SP は、各ノードにおいて解のまとめ上げを行うプロセスである。

DM は、RP,SP が処理を行う際に必要なデータを管理するプロセスで、ノードが保持する情報の管理と、ルックアップ処理の待ち合わせ機構を実現している。

IP の各モジュール構成を下図 6.2–1 に示する。図中丸印で書かれたものはプロセスをあらわし、四角で書かれたものはサブルーチンをあらわしている。

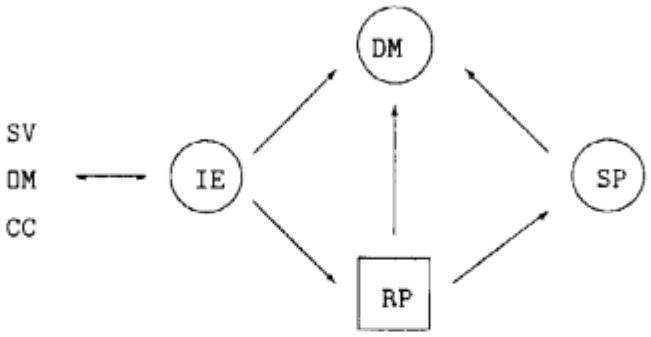


図 6.2-1 IP モジュール構成図

```
&b_pgm;;
    &b_obj;; int >= even;; &e_obj;;
    &b_rule;;
        m::p[1=5] <= o/[1->even];;
        m::p[1=8]/[1->int] <= o/[1->odd];;
        m::o/[1->int];;
    &e_rule;;
&e_pgm.
```

図 6.3-1 解の説明のための例題

6.3 アルゴリズム

本項では、まず *Quixote* の解について説明し、続いて、本インタプリタの推論処理部分で採用されているアルゴリズムについて概要を述べる。

① *Quixote* の解

Quixote の解は、OM からの interpret メッセージ中で指定されたゴール節に含まれる変数に対する束縛の集合 B と、その束縛を得る際に用いられた仮定 (assumption) の集合 A の組 $A \Leftarrow B$ である。

ルールのボディ部 (または問い合わせ中) に現れるドット項に関する制約が、データベースの記述と矛盾はしないが、その帰結でもない場合、その制約は仮定とされる。インタプリタの動作は、仮定が満たされたものとして続行されるが、それによって得られた解には、その解を得るために用いられた仮定が付加される。

例を用いて説明する。図 6.3-1 のデータベースに対して、`?- m:p[1=X]/[1 -> int].` と

いう問い合わせを発した場合を考える。o!1 の値に関して、データベースは int より小さいと言っているが、even ないし odd より小さいとまでは言っていない。しかし、その一方とは矛盾しない。この場合、各ルールは、それぞれ o!1 =< even, o!1 =< odd を仮定して成功する。同様の現象が、問い合わせ中の p[1=X]/[1 -> int] と、帰結 m:p[1=5] との間にも生じる。問題となっているラベルに対して何の言及もない帰結に関しては、&top よりも小さいと主張していると考えることにする。この結果、解は次の二種類となる。

$$\{\texttt{o!1 =< even, p!1 =< int}\} \Rightarrow \{\texttt{X == 5}\},$$
$$\{\texttt{o!1 =< odd}\} \Rightarrow \{\texttt{X == 8}\}$$

② 導出木の生成

導出木のデータ構造、生成アルゴリズムの概要については、次節 (②) で RP サブモジュールの機能の概要とともに述べる。

③ 解のまとめ上げ

プロローグと異なり、*Quixotε* では、同じオブジェクトに関する解を統合せねばならないので、生成した導出木から得られる制約を各オブジェクトごと計算し、同じ仮定の元で解を導いているのであれば、それらをまとめて、ゴール節の変数に対する制約と、その際の必要となる仮定を再計算しなくてはならない。この処理を**解のまとめ上げ**と呼ぶ。

制約の集合 S_1, S_2 について、S_1 が S_2 に**包摂される**とは、任意の S_1 の元 s に対して S_2 のある部分集合 S_2' が対応しており、S_2' の元が全て成り立っているとき s も成り立つことが示されることをいい、$S_1 \sqsubseteq_S S_2$ と書く。$S_1 \sqsubseteq_S S_2 \ \text{—}\ S_2 \sqsupseteq S_1$ であり、$S_1 \sqsubseteq_S S_2 \wedge S_2 \sqsubseteq_S S_1 \ \text{—}\ S_1 \cong_S S_2$ である。

また、制約の集合 S が**矛盾する**とは、ある変数またはドット項があって、S 中で解を持たないことが示されることをいい、$null(S)$ で表す。

任意の二解 R1=(A1 ⇒ B1), R2=(A2 ⇒ B2) があったとき、これらをまとめる際のアルゴリズムを、(言語風に、図 6.3-2 に示す。

6.4 各モジュールの機能概要

以下では、各サブモジュールの機能と実装について、概要を述べる。

① IE

IE は、外部モジュールとの窓口となるプロセスである。以下の機能がある。

(a) IP の起動

IP を起動し初期化する。IP に要求がくると SV,OM,CC の外部ストリームを受けとり保持する。DM プロセスを生成し初期化する。初期化の結果を待ち合わせ正常に終了したならば、プロセスループを開始する。

```
merge2Result(R1=(A1 ⇒ B1), R2=(A2 ⇒ B2))
{
  switch (check_assumption_relation(A1,A2)) {
    case A1 ≅S A2:  % (同一仮定の場合)
      return R=(A1 ⇒ B1 ∪ B2);
    case A1 ⊑S A2:  % (一方が包摂する場合)
      if (B2 ⊑S B1)
        return R1;
      else
        return R=(A1 ⇒ B1 ∪ B2), R2;
    case !null(A1 ∪ A2):  % (共通部分がある場合)
      return R1, R2, R=(A1 ∪ A2 ⇒ B1 ∪ B2);
    default:
      return R1, R2;
  }
}
```

図 6.3-2 二解をまとめるアルゴリズム

(b) トランザクション処理

トランザクションの開始 / 終了 / 中止に伴うトランザクションの状態の変更と保持を行う。

(c) DB セッション処理

DB セッションの開始 / 終了の状態を保持する。ただし、DB セッションが終了しても IE プロセスは自滅せずに次の要求が来るまで待機状態を継続する。

(d) 推論開始処理

問いが発せられて OM から推論開始のメッセージ (interpret) を受けとると DM を初期化してから RP に展開処理の開始を指示する。

(e) 終結

IP は、問い合わせや DB セッション終了後も *Quixote* がシャットダウンされるまで死滅することはなく、待機状態として生存している。

SV からの終結メッセージ (kill) を受けとると DM プロセスの終結を促し、IP が保持している外部ストリーム (SV,OM,CC) をすべて閉じて終結する。

② RP

(a) 概要

RP は、与えられた問合せに対してルール / ファクトを適用して導出木を作りあげるサブルーチンである。その際の特徴としてルックアップ機能がある。ルックアップ機能はサブゴールノードを展開する際に別のサブゴールノードの展開結果を参照して済ませる機能である。以下では RP がつくる導出木に関する説明とルールを取り出す際の条件、ルックアップする際の条件について記述する。

(b) 導出木

RP が展開する導出木の説明をする。導出木は、一種の n 進 AND-OR 木である。ノードにはサブゴールノードとルールノードの 2 種類ある。サブゴールノードはユニファイ可能なルールを子供のルールノードとして持つ OR ノードである。ルールノードは、ルールのボディ部のサブゴールを子供のノードとして持つ AND ノードである。したがって、導出木はサブゴールノードとルールノードが交互に繰り返し現れる構造になっている。

ルートノードは特別なオブジェクト項を持つサブゴールノードとなっている。ルートノードのオブジェクト項と、ルートノードの直下の子ノードに当たるルールノードのルールは一回の問合せごとに一時的につくられる。このルールはボディ部として問合せ列を持つような構造をしている。

導出木の各ノードはノードプロセスとして実現されている。ノードの展開が終り導出木ができるとリーフに当たるノードプロセスから解併合が開始される。

また、各ノードに付随する詳細な情報は DM で管理している。

(c) ルックアップ条件

サブゴールノードではルックアップ可能なノードを検索してあればルックアップによる解併合を開始するように支持してそのノードの以降の展開を終了する。この際に次のような条件を検査する。サブゴールノードを展開する際に既に展開中か、あるいは、展開されているサブゴールがあるかどうかを調べる。現時点で展開されている各サブゴールについて同一モジュールないでなおかつオブジェクト項部分が同値な物をルックアップ可能とする。

(d) ルール取得条件

ルックアップ条件を満たすノードがない場合は、ルールを取得してルールノードの展開を続ける。ルールヘッドのオブジェクト項部分がサブゴールのオブジェクト項部分とユニファイ可能なルールを取り出す。また、継承も考慮するように継承モードが指定されているときは、ユニファイ可能なルールに加えてサブゴールのオブジェクト項部分とルールヘッドのオブジェクト項部分に包摂関係があるルールを取り出す。

(e) 展開アルゴリズム

RP でおこなう導出木の展開処理を簡単に定式化する。

```
expand(SG)
```

```
BEGIN
SG := Root                      % ルートノードのオブジェクトをサブゴール SG とおく
create_subgoal_node(SG)         % サブゴールノードを作成する
if check_lookup(SG,LG)==yes     % ルックアップ可能ならば、
    then send(lookup(LG))       % lookup メッセージを送って終了
         exit
get_rule(SG,Rules)              % SG と単一化可能なルールを得る
if Rules==[]                    % なければ、
    then send(fail)             % fail メッセージを送って終了
         exit
for each Rule of Rules          % それぞれのルールについて
    create_rule_node(Rule)      % ルールノードを作成する
    if check_body(Rule)==[]     % ボディ部が空 ([]) ならば、
        then send(success)      % success メッセージを送って終了
             exit
    for each SG of Body         % ボディ部のサブゴールそれぞれについて
        expand(SG)              % 再帰的に展開する
END
```

以下に、上記アルゴリズムを説明する。

(i) ルートノードをサブゴールとおく。

(ii) サブゴールノードに関する情報を DM に登録し、サブゴールノードに対応するサブゴールノードプロセスを生成する。

(iii) ルックアップ条件を検査する。

(iv) ルックアップ可能であれば、サブゴールノードプロセスにルックアップによる解併合を開始するように支持し、そのノードの以降の展開を終了する。

(v) ルックアップ不可能であれば、ルール取得条件に適合するルールを収集する。

(vi) 条件に合致するルールが一つもなければそのノードは失敗ノードであり、サブゴールノードプロセスに失敗した旨の通知をして解併合を開始するように支持し、そのノードの以降の展開を終了する。

(vii) 条件に合致したルールが一つでもあればそれぞれのルールについてルールに関する情報を DM に登録し、ルールノードに対応するルールノードプロセスを生成する。

(viii) ルールのボディ部を調べてサブゴールがなければそのルールはファクトであり、そのノードは成功ノードであるので以降の展開を終了し、ルールノードプロセスに成功した旨の通知をして解併合を開始するように支持する。

(ix) ボディ部に一つでもサブゴールがあれば、それぞれのサブゴールについて上記(ii)からのサブゴールの展開を繰り返し行う。

③ SP

SP は、展開された導出木に散らばった情報から解をまとめ上げるプロセスである。以下の機能がある。

(a) **ノードでの解のまとめ上げ**

RP によって導出木のノード上で起動されるプロセスを**ノードプロセス**という。ノードプロセスは、子ノードのノードプロセスから解を受けとり、自身の解を計算して親ノードのノードプロセスに送信する。サブゴールノードのノードプロセスを**サブゴールノードプロセス**といい、ルールノードのノードプロセスを**ルールノードプロセス**という。

ノードプロセスは、起動されると、RP が DM にセットしておいた各種のデータのうち、解のまとめ上げに必要なものを取り出す。子ノードから解が上がってくるのを待ち、すべて揃ったら、親ノードに渡す形に加工して、DM の定められた場所にセットする。

(b) **ルックアップによる解のまとめ上げ**

サブゴールノードが RP から rp_lookup メッセージを受けとると、ルックアップ処理を行う。ルックアップの際には、通常と異なる手順で解を求める。

- **祖先以外のノードのルックアップ**

  DM の機構を用いて、参照先の解が求まるまで待ち合わせをする。参照先の展開が失敗した場合、自身の展開も失敗させる。参照先が成功したならば、その解を自身の解として複写してくる。

- **祖先ノードのルックアップ**

  祖先ノードのルックアップの際には、特別のアルゴリズムが用いられる。通常、各ノードプロセスは、子ノードから解を得、それらをまとめて親ノードに渡せば良いのだが、祖先ノードをルックアップしている場合には、ルックアップによって解を複写してきたことによって自身の解が、ひいてはその祖先ノードの解が書き変わることがあるので、一度の参照では終了しないことがある。そこで、部分解を参照し、解を再計算し、再び部分解を参照するという繰り返しとなる。全解の部分解から新しい解が得られていない状態になると、繰り返しが終了し、解が確定する。

  ルックアップしている葉のサブゴールノードから、ルックアップされているサブゴールノードまでの経路上の各ノードプロセスは同期して動作する。これらのノードを**ルックアップ木**をいう。これらのプロセスの同期は、**トリガー変数**と呼ばれる KL1 変数によって行う。ルックアップ木の頂点ノードのノードプロセスは、新しい解が

得られたときや、導出の失敗 / 成功が判明したときにトリガー変数をバインドし、新しい解の再計算 / 解の登録 / 失敗を指示する。

(c) 推論結果の生成

導出木中のデータ構造は、外部にそのまま出すことのできない内部表現であるので、SP には、根ノードの解の内部表現から OM に返すことのできる解を生成するフィルターが用意されている。

④ DM

DM の主な機能としては IP で使用するデータの管理とルックアップ処理のための条件判定とがある。

(a) データ管理機能

DM では展開処理で生成される導出木に関する情報を管理している。これらの情報は展開時に動的に蓄えられ、参照される。

導出木の情報は各ノード毎にノード識別子を割り振り、ノード毎に管理する。DM では、ノードの生成、ノード情報の格納 / 取り出しなどの操作をノード識別子を介したプロトコルとして提供している。

(b) ルックアップ処理

ルックアップとは、あるサブゴールノードの解を求める際に、実際にそのノード以下の展開をせずに、そのノードと同じ導出木の展開が得られる他のノードの解を参照して済ませる処理である。

この処理の目的のひとつは、過去の計算結果をできるだけ有効に使って計算効率を上げることにあり、もうひとつは、再帰的に同じサブゴールが呼ばれている場合で本来停止すべきループを検出し無限ループに陥るのを防ぐことにある。

DM では、ルックアップ処理を可能にするためにルックアップ可能かどうかを調べるルックアップ条件検査とルックアップ先の解を参照する際の参照待ち合わせの機能がある。

(i) ルックアップ条件検査

RP がサブゴールノードを展開する際に、既に生成されたサブゴールノードの中でルックアップできるノードがあるかどうかを調べる。

展開中のサブゴールと既に展開されたサブゴールとの間で以下の条件を満たすかどうかを調べる。

- サブゴールが同一モジュール内にあり、かつ、サブゴールのオブジェクト項部分が同値である。
- 相互参照状態になっていない。

ここで、相互参照とは、ルックアップ先の子孫ノードがルックアップ元の先祖ノードをルックアップしている状態のことをいう。相互参照の状態にあると、相互に相互の解を得よ

うと待ち続けることになり、解が求まらなくなってしまうので、この場合、ルックアップ不可能として通常のサブゴール展開をする。

(ii) 参照待ち合わせ

各ノードプロセスは並列的に処理しているため、ルックアップによってノードの解併合する際には、ルックアップ先のノードの解併合が終了していなければならない。

DM は SP からの問い合わせに対して自身で管理しているノード情報をしらべ、必要に応じてルックアップ先の解併合を待ち合わせてから、ルックアップ元のノードに解の参照を許可する。

7 制約コンパレータ (CC)

7.1 概要

Quixote の CC モジュールは、制約解消系を中心として、オブジェクト項の間の操作を行うサブルーチンの集まりである。これらは主としてインタプリタにて用いられる。CC は大別すると次のように分類される。

1. CC プロセス

   プロセスの作成

2. オブジェクト項間の操作

   単一化、マッチャ、同一性判定、ミート、ジョイン、包摂関係

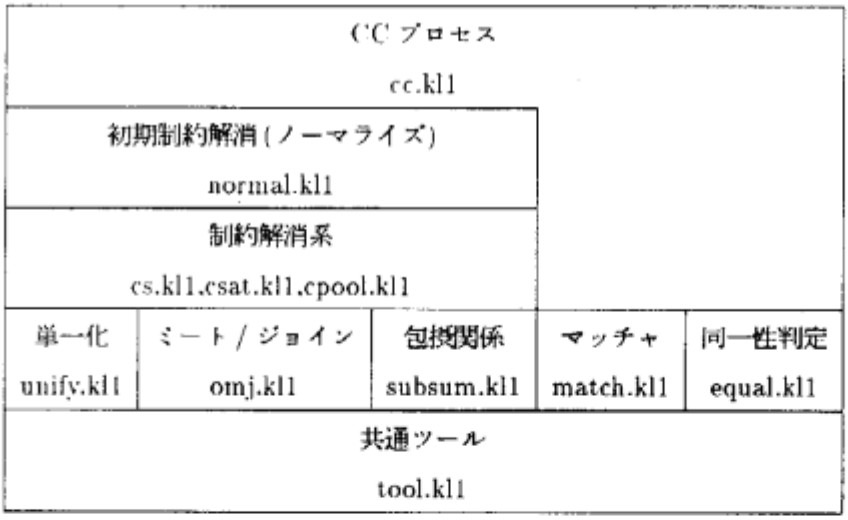
3. 制約解消系

   制約解消、プログラムの初期制約解消

4. CC 共通ツール

7.2 モジュール構成

Quixote CC 内部のモジュール構成図は以下の通りである。



① cc プロセス (cc.kl1)

CC の各サブルーチンは、CC モジュールへのメッセージとしても実行できる。CC プロセスの作成及びメッセージの処理を行うのが本モジュールである。

② 単一化 (unify.kl1)

オブジェクト項同士の単一化を行う。現実装では、単一化は unsafe-unification を採用している。

③ マッチャ (match.kl1)

オブジェクト項同士のマッチャ (variant) である。二つのオブジェクト項を同一にする変数の置換を求める。

④ オブジェクト項の同一性判定 (equal.kl1)

変数のバインドを考慮して二つのオブジェクト項の同一性を判定する。

⑤ 基底オブジェクト項のミート、ジョイン (omj.kl1)

二つのオブジェクト項のミート、ジョインを計算する。

⑥ オブジェクト項の subsume(subsum.kl1)

二つのオブジェクト項の包摂関係を求める。その他、SD からのメッセージを処理する。

⑦ プログラムの初期制約の解消: ノーマライズ (normal.kl1)

QUIXOTE のプログラムに含まれる初期制約の解消を行う。変換 3 にて使われる。

⑧ 制約解消系 (cpool.kl1, cs.kl1, csat.kl1)

制約解消を行う。制約を貯めるプールプロセスと、個々の制約を変換する制約変換系から成る。

⑨ 共通ツール (tool.kl1)

CC モジュールで使われる各種サブルーチン集

7.3 制約解消アルゴリズム

ここでは、*QUIXOTE* の CC モジュールの内制約解消のアルゴリズムを述べる。前述のように制約解消系は複数のファイルから構成される。それらの主な役割は以下である。

normal.kl1 初期制約解消 (ノーマライズ) エントリ

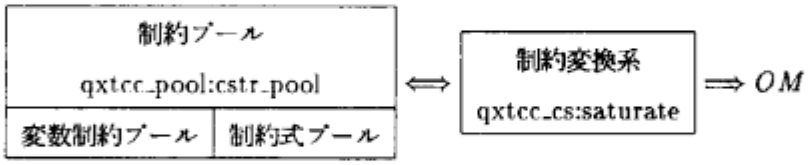
cs.kl1 制約解消系エントリ、ツール

cpool.kl1 制約プール

csat.kl1 制約変換系

① 動作概要

制約解消系の処理は次のように制約プールと制約変換系の二つのプロセスの間の通信を通じて行われる。



*QUIXOTE*CC の制約解消の処理は以下のように定式化されるであろう。

```
loop
        get(Cold) % 制約プールから制約式を一つ取る
        if (Cold==[]) return(yes)       % 制約プールが空ならば成功
```

```
        process(Cold,Cnew) % 制約変換系にて制約式を簡約する
        if inconsistent(Cnew) return(no)    % 簡約で矛盾があれば失敗
        add(Cnew) % 簡約された制約を制約プールに加える
    loop_end
```

② 制約プール

まだ満たされていない制約を格納する制約式プールと、変数制約を格納する変数プールからなる。制約式には次の3つの形がある。

バインディング制約: {T,U} ただし T はオブジェクト項, U は変数。

等号制約: {T,==,U} ただし T,U はオブジェクト項

包含制約: {T,=<,U} ただし T,U はオブジェクト項

変数プールは、一度 OM から読み込んだ変数の制約を溜めておくキャッシュであり keyed-set により実現されている。制約変換途中の制約の書き換えは、この変数プールに対して行われる。これによりスピードアップとともに、制約変換失敗時にオリジナルデータを破壊しないという利点がある。

制約式プールでは次のように4つのレベルに分けて制約を格納している。制約プールのプロトコル put(^Cstr) は最も低いレベルの制約式を一つ制約式プールから取り出し、Cstr にバインドする。取り出された制約式は制約式プールから消去される。これは制約変換で失敗を早く検出するためのヒューリスティックである。

レベル 0 {O,O},{O,==,O},{O,=<,O}

レベル 1 {G,V} G は基底オブジェクト項, {T1,==,T2},{G1,=<,G2} G1,G2 は基底オブジェクト項

レベル 2 {O1,=<,O2} O1,O2 はオブジェクト項, {O,V} O はパラメトリックオブジェクト項

レベル 3 {U,V} U は変数, {V1,=<,V2} V1,V2 は変数

③ 制約変換系

制約プールから制約を一つ取りだし、OM と通信を行いながらそれを簡約し、簡約したものを再び制約プールに格納する。主な制約の処理を以下に示す。

- レベル 0 の制約

  トートロジーなのでなにもせずに消去

- {G,V} G は基底オブジェクト項

  1. $\forall s \in$ V.*add.small*, {s,=<,G}
  2. $\forall l \in$ V.*add.large*, {G,=<,l}

- {T1,==,T2}
  1. T1 または T2 が変数の場合には、 ref を張りバインディング制約({T1,T2} または {T2,T1}) に帰着
  2. T1,T2 がオブジェクト項の場合には、対応するヘッド、属性値の等号制約に分解
- {G1,=<,G2} G1,G2 は基底オブジェクト項
  qxtcc_sbs:subsume を呼んでチェックする。
- {O1,=<,O2} O1,O2 はオブジェクト項
  対応するヘッド、属性の包含制約に分解
- {U,V} U は変数
  1. $\forall$s $\in$ V.*add.small*, $\forall$l $\in$ U.*add.large*, {s, =<, l}
  2. $\forall$s $\in$ U.*add.small*, $\forall$l $\in$ V.*add.large*, {s, =<, l}
  3. V.add.small に U.add.small をマージする
  4. V.add.large に U.add.large をマージする
- {V1,=<,V2} V1,V2 は変数
  1. $\forall$s $\in$ V1.*add.small*, $\forall$l $\in$ V2.*add.large*, {s, =<, l}
  2. V2.add.small に V1 および V1.add.small をマージする
  3. V1.add.large に V2 および V2.add.large をマージする

7.4 サブモジュールの機能概要

ここでは CC 内の各サブモジュールのうち代表的なサブルーチンを紹介する。

① CC モジュール

qxtcc:create(ToSV,ToOM,ToIP,InCC,Ŝtatus) CC プロセスの起動。

ToSV,ToOM,ToIP は他モジュールへのストリーム、**InCC** は CC に入ってくるストリーム。

② オブジェクト項間の操作

qxtcc_uni:unifycs(oldStream,newStream,Tid,T1,T2,B̂t,R̂es,Ŝtatus) オブジェクト項を単一化し、その単一化子を用いて制約解消を行う。

oldStream,newStream OM へのストリーム (oldnew 型)

Tid トランザクション ID

T1,T2 オブジェクト項

Bt 単一化子 ({bind 先, 変数} のリスト)

Res yes,no, または error

Status エラー情報 normal または abnormal({ メッセージ })

qxtcc_equ:termequal(oldStream,newStream,Tid,T1,T2,$\hat{\text{R}}$es,$\hat{\text{S}}$tatus) オブジェクト項の変数環境を考慮した同一性を調べる。

T1,T2 オブジェクト項

Res T1 と T2 が同一の場合 yes, そうでなければ no, エラー時は error

Status エラー情報 normal または abnormal({ メッセージ })

qxtcc_mat:termmatch(oldStream,newStream,Tid,T1,T2,$\hat{\text{B}}$t,$\hat{\text{R}}$es,$\hat{\text{S}}$tatus) マッチャ (variant) を求める。

T1,T2 オブジェクト項

Bt T1,T2 を等しくする変数の一対一対応関係 ({T1 の変数,T2 の変数 } のリスト)

Res yes,no, または error

Status エラー情報 normal または abnormal({ メッセージ })

qxtcc_sbs:subsume(oldStream,newStream,Tid,T1,T2,$\hat{\text{R}}$es,$\hat{\text{S}}$tatus) オブジェクト項の包含関係を求める。

T1,T2 オブジェクト項

Res (T1 $\sqsupseteq$ T2) の場合に yes, そうでなければ no, エラー時は error

Status エラー情報 normal または abnormal({ メッセージ })

ここで、T1,T2 が基底オブジェクト項の場合には以下の操作を行う。

1. T1, T2 が基礎オブジェクト項の場合
   基礎オブジェクト項ラティスに問い合わせる
2. T1= $o[l_1 = v_1, \ldots, l_n = v_n]$, T2=$p[m_1 = u_1, \ldots, m_k = u_k]$ の場合
   T1 $\sqsupseteq$ T2 $\Longleftrightarrow$ $o \sqsupseteq p \wedge \forall i \in \{1, \ldots, k\}, \exists j \in \{1, \ldots, n\}, m_k = l_j, v_k \sqsupseteq u_k$

また、T1,T2 のいずれかが変数の場合には次のように判定する。

- T1,T2 ともに変数の場合
  次のいずれかの場合に限り Res が yes となる。
  1. T1 の add:small に T2 が含まれている
  2. T2 の add:large に T1 が含まれている
  3. T1 の add:small に基底項 G1、T2 の add:large に基底項 G2 が存在し、G1 $\sqsupseteq$ G2 が成り立つ
- T1 変数, T2 オブジェクト項の場合
  次のいずれかの場合に限り Res が yes となる。

1. T1 の add:small に T2 が含まれている
2. T1 の add:small に基底項 G1 が存在し、G1 ⊒ T2 が成り立つ

- T1 オブジェクト項, T2 変数の場合

  次のいずれかの場合に限り Res が yes となる。

1. T2 の add:large に T1 が含まれている
2. T2 の add:large に基底項 G2 が存在し、T1 ⊒ G2 が成り立つ

qxtcc_omj:omeet(oldStream,newStream,Tid,T1,T2,Meet,Status) オブジェクト項のミートを求める

T1,T2 基底オブジェクト項

Meet T1 と T2 のミート、エラー時は error

Status エラー情報 normal または abnormal({ メッセージ })

qxtcc_omj:ojoin(oldStream,newStream,Tid,T1,T2,Join,Status) オブジェクト項のジョインを求める

T1,T2 基底オブジェクト項

Join T1 と T2 のジョイン、エラー時は error

Status エラー情報 normal または abnormal({ メッセージ })

③ 制約解消系

qxtcc_cs:cs(oldStream,newStream,Tid,Cstr,Bt,Res,Status) 制約解消系

Cstr 制約のリスト。個々の制約式は、{T1,==,T2},{T1,=<,T2},{T1,T2} のいずれかの形をしている。

Bt 単一化子 (失敗時は制約変換の履歴)

Res yes,no, または error

Status エラー情報 normal または abnormal({ メッセージ })

Res が yes の場合にのみ変数ブールの情報を OM に戻す。それ以外の場合にはオリジナルの変数制約データを破壊しない。

qxtcc_nor:normalize(oldStream,newStream,Tid,Cstr,Remain,Bt,Res,Status) ノーマライズ (初期制約解消)。Remain は未解決ドット項制約のリスト

④ CC 共通ツール

qxtcc_uni:decompose_oterm(oldStream,newStream,Tid,O,Head,Attr,Status) オブジェクト項を分解する。オブジェクト項間操作のサブルーチンで共通に使われる。

O オブジェクト項

Head O のヘッド (bobj)

Attr O の属性値対のリスト

Status エラー情報 normal または abnormal({ メッセージ })

(付録) 構文、提供機能

概要

Quixotes は、ESP あるいは KL1 で書かれたユーザ・プログラムから使用することができる。7.4 で、KL1 プログラムから呼び出すことのできるメッセージの概要を示し、7.4 では、問合せの構文を示す。

次節以降の構文の記法は以下の通りである。

① 基本的に BNF を用いる。ただし以下の記法も使っている:

[···]	··· は省略可能
{···}	··· が 0 回以上
<···>-*list*	<···> {<···>} ; ··· が 1 回以上
<···>-*list*("a")	<···> {"a" <···>} ; ··· がデリミタ "a" 付きで 1 回以上
a (*a'*)	*a* は論文用、*a'* は実際の記法

② コメントは "%" から行末 (CR) までとする。

提供機能

① ストリームの作成:

qxtIF:create(User_stream, Qxt_stream) — モジュール qxtIF への述語呼出し

② DB との接続、DB 単位の操作:

```
create_database(DB_name,Lib,DB_content, Status)
open_database(DB_name,Version,Lib(s),OpMode, Status)
open_databases([{DB_name,Version,Lib(s),OpMode},···], Status)
delete_database( Status)
delete_databases([{DB_name,Version},···], Status)
close_database(Lib,ClMode,Exit, Status)
close_databases([{DB_name,Version,Lib,ClMode},···],Exit, Status)
begin_transaction( Status)
begin_transaction({DB_name,Version}, Status)
end transaction( Status)
end_transaction({DB_name,Version}, Status)
abort_transaction( Status)
abort_transaction({DB_name,Version}, Status)
```

③ モジュール単位の操作:

```
show_module( Mod_Hierarchy_stream, Status)
show_module({DB_name,Version}, Mod_Hierarchy_stream, Status)
show_module_nodes(Mid,Neighbor, Mod_stream, Status)
show_module_nodes({DB_name,Version}, Mid,Neighbor, Mod_Hierarchy_stream, Status)
        % submodule 関係をストリームで返す
create_module(Mod_Id,Wherels,Rule_stream, Status)
create_module({DB_name,Version}, Mod_Id,Wherels,Rule_stream, Status)
    % Wherels の場所に Rule_stream からなる Mod_Id をもつモジュールを作成する
recreate_module(Mod_Id,AbsDel_stream, NewMod_Id, Status)
recreate_module({DB_name,Version}, Mod_Id,AbsDel_stream, NewMod_Id, Status)
    % Mod_Id (のモジュール) に AbsDel_stream を順次適用し、NewMod_Id を作成
delete_module(Mod_Id, Status)
delete_module({DB_name,Version},Mod_Id, Status)
modify_submodule_link(ModLinks, Status)
modify_submodule_link({DB_name,Version},ModLinks, Status)
        % submodule 関係に ModLinks の内容を追加する。
modify_module_link(Link_name,ModLinks, Status)
modify_module_link({DB_name,Version}, Link_name,ModLinks, Status)
        % Link_name をもつリンクに ModLinks の内容を追加する
```

④ オブジェクトの操作:

```
show_lattice(Object, Object_Hierarchy_stream, Status)
show_lattice({DB_name,Version},Object, Object_Hierarchy_stream, Status)
show_lattice_nodes(Object,Neighbor, Object_stream, Status)
show_lattice_nodes({DB_name,Version},Object,Neighbor, Object_stream, Status)
        % subsumption 関係をストリームで返す。
modify_lattice(Object,Links, Status)
modify_lattice({DB_name,Version},Object,Links, Status)
        % Object に subsumption 関係の Links を付加する
modify_subsumption_link(ObjLinks, Status)
        % subsumption 関係に ObjLinks を付加する
modify_object_link(Link_name,ObjLinks, Status)
modify_object_link({DB_name,Version},Link_name,ObjLinks, Status)
        % Linkname をもつリンク関係に ObjLinks を付加する
compress_set(Object_stream,Compress_Para, New_Object_stream, Status)
```

⑤ ルールの操作:

```
show_rules(Mod_Id, Rule_stream, Status)
show_rules({DB_name,Version},Mod_Id, Rule_stream, Status)
        % Mod_Id に含まれているルールをストリームで返す
show_objects(Mod_Id, Supported_Object_stream,Status)
show_objects({DB_name,Version}, Mod_Id, Supported_Object_stream, Status)
        % Mod_Id でサポートされているオブジェクトをストリームで返す
insert_rule(Mod_Id,Rules, Status)
insert_rule({DB_name,Version},Mod_Id,Rules, Status)
        % Mod_Id に Rules を追加する
```

⑥ 問合せ:

```
query(Query, Answer, Status)
query({DB_name,Version},Query, Answer, Status)
```

構文

① プログラム

```
<program>       ::=  <program-def> "."
<program-def>   ::=  <b-pgm> ";;" <section>-list(";;") ";;" <e-pgm>
                   | <section>-list(";;")
<section>       ::=  <env-sect>
                   | <obj-sect>
                   | <mod-sect>
                   | <rule-sect>
<b-pgm>         ::=  "&begin_program" | "&b_pgm"
<e-pgm>         ::=  "&end_program"   | "&e_pgm"
```

② 環境部

```
<env-sect>      ::=  <b-env> ";;" <env>-list(";;") ";;" <e-env>
<env>           ::=  "&name[&pgm-name=" <pgm-name>"]"
                   | "&author[&aut-name=" <aut-name>"]"
                   | "&date[&date="<date>"]"
                   | <b-exp> ";;" <exp-def>-list(";;") ";;" <e-exp>
                   | "&include""["<def-lib>-list(",")"]"
<exp-def>       ::=  <exp-name> "=" <o-term>
                   | <exp-name> "=" <m-id>
                   | <exp-name> "=" "&del" "("<lab>-list(",")")" <exp-name>
                   | <exp-name> "=" <exp-name>["["<o-attr>-list(",")"]"
                                                ["|""{"<o-cnstr>-list(",")"}"]]
                   | <exp-name> "=" "&abs"
                     "(""["<o-attr>"]" ["|""{"<o-cnstr>-list(",")"}"]")"<exp-name>
<def-lib>       ::=  <lib-lab> "=" <lib-name>
                   | <lib-lab>+ (<lib-lab>"+") "=" "{"<lib-name>-list(",")"}"
                   | "&sort_lib=" <sort-mod>
<b-env>         ::=  "&begin_environment_section" | "&b_env"
<e-env>         ::=  "&end_environment_section"   | "&e_env"
<b-exp>         ::=  "&begin_expression" | "&b_exp"
<e-exp>         ::=  "&end_expression"   | "&e_exp"
<pgm-name>      ::=  <string>
<aut-name>      ::=  <string>
<date>          ::=  <string>
<exp-name>      ::=  "e_"<string>
<lib-name>      ::=  <string>-list("/")
<lib-lab>       ::=  "&exp_lib" | "&pgm_lib"
<sort-mod>      ::=  <l-string>
```

③ モジュール間関係部

<mod-sect> ::= <*b-mod*> ";;" <mod-map>-*list*(";;") ";;" <*e-mod*>
<mod-map> ::= <*mod-map*> ";;" <*submod*> ";;" <m-sub>-*list*(";;")
| <*mod-map*> ";;" <*link-name*> ";;" <m-link>-*list*(";;")
<m-sub> ::= <m-id> "⊒$_S$" (">-") <m-desc>
<m-desc> ::= <m-id>
| "(" <m-desc> ")"
| <m-desc> "∪" ("+") <m-desc>
| <m-desc> "\" ("-") <m-desc>
<m-link> ::= <m-id>"⟶"("-->") <m-id>
| <m-id>"⟶"("-->")"{"<m-id>-*list*(",")"}"
<*b-mod*> ::= "**&begin_module_section**" | "**&b_mod**"
<*e-mod*> ::= "**&end_module_section**" | "**&e_mod**"
<*mod-map*> ::= "**&module_map**" | "**&mod_map**"
<*submod*> ::= "**&submodule**" | "**&submod**"
<*link-name*> ::= <*l-string*>

④ オブジェクト間関係部

<obj-sect> ::= <*b-obj*> ";;" <obj-map>-*list*(";;") ";;" <*e-obj*>
<obj-map> ::= <*obj-map*> ";;" <*subsum*> ";;" <obj-sub>-*list*(";;")
| <*obj-map*> ";;" <*link-name*> ";;" <obj-link>-*list*(";;")
<obj-sub> ::= <*basic-obj*> <*sub-rel*> <*basic-obj*>
| <*basic-obj*> <*sub-rel*> "{"<*basic-obj*>-*list*(",")"}"
<obj-link> ::= <o-term> "⟹" ("==>") <o-term>
| <o-term> "⟹" ("==>") "{"<o-term>-*list*(",")"}"
<*b-obj*> ::= "**&begin_object_section**" | "**&b_obj**"
<*e-obj*> ::= "**&end_object_section**" | "**&e_obj**"
<*subsum*> ::= "**&subsumption**" | "**&subsum**"
<*obj-map*> ::= "**&object_map**" | "**&obj_map**"
<*sub-rel*> ::= "⊑" ("=<") | "⊒" (">=") | "≅" ("==")

⑤ ルール定義部

<rule-sect> ::= <*b-rule*> ";;" <rules> ";;" <*e-rule*>
<*b-rule*> ::= "**&begin_rule_section**" | "**&b_rule**"
<*e-rule*> ::= "**&end_rule_section**" | "**&e_rule**"
<rules> ::= <rule>
| <rules> ";;" <rules>

```
% 一般のルール
<rule>      ::=  [<m-lab> "::"] <clause>
               | [<m-lab> "::"] "{{"<clause>-list(";;") "}}"
<clause>    ::=  <rule-a-term> ["⇐" ("<=") <body>]
<body>      ::=  <props>["||" "{"<r-cnstr>-list(",")"}"]
<props>     ::=  <prop>
               | <props> "," <props>
<prop>      ::=  [<m-id>":"] <a-term>
               | <a-cnstr>
<m-lab>     ::=  <m-id>
               | "{"<m-id>-list(",")"}"
<r-cnstr>   ::=  <a-cnstr>

% <query> の問合せ用
<query-body>   ::=  <query-props>["||""{"<r-cnstr>-list(",")"}"]
<query-props>  ::=  <query-prop>
                  | <query-props> "," <query-props>
<query-prop>   ::=  [<m-id>":"] <query-a-term>
                  | <a-cnstr>
```

⑥ 属性項

```
<a-term>      ::=  <o-term> [<sub-rel> <objs>]
                       ["/"["["<a-attr>-list(",")"]"] ["|" "{"<a-cnstr>-list(",")"}"]]
<objs>        ::=  <o-term>
                 | "{"<o-term>-list(",")"}"
<a-attr>      ::=  <lab> <attr-op> <ind-term>
                 | <set-lab> <attr-op> <set-terms>
                 | <lab> "→" ("->") <set-terms>
                 | <set-lab> "←" ("<-") <ind-term>
<ind-term>    ::=  <o-term>
                 | <m-id>
                 | <sort>
<a-cnstr>     ::=  [<m-id>":"] <ind-term> <sub-rel> [<m-id>":"] <ind-term>
                 | [<m-id>":"] <set-lhs> <set-rel> [<m-id>":"] <set-terms>
<sort>        ::=  <sort-name>["/""["<a-attr>-list(",")"]"]
                                          % 現在は <sort-name> しか許されない。
<sort-name>   ::=  "s_"<c-o-term>

% 集合用
<set-lab>     ::=  <lab>"+" (<lab>"+")
<set-terms>   ::=  "{"<o-term>-list(",")"}" % 現在は <g-var>,<set-var> は含まない。
                 | "{"<m-id>-list(",")"}"   % 現在は <g-var>,<set-var> は含まない。
                 | <g-var>
                 | <o-term>"!"<set-lab>
                 | <set-var>
<set-lhs>     ::=  <o-term>"!"<set-lab>
                 | <set-var>
<set-var>     ::=  <g-var>"+" (<g-var>"+")
```

% <rule> のヘッド用

<rule-a-term> ::= <o-term> ["/"["["<a-attr>-*list*(",")"]"]
["|""{"<a-cnstr>-*list*(",")"}"]]

% <query> の問合せ用

<query-a-term> ::= <o-term>[<*sub-rel*><objs>] ["/"["["<query-a-attr>-*list*(",")"]"]
["|""{"<a-cnstr>-*list*(",")"}"]]

<query-a-attr> ::= <a-attr>
| "{"<lab>"}" <*attr-op*> <ind-term>
| "{"<set-lab>"}" <*attr-op*> <set-terms>

<*set-rel*> ::= "$\sqsubseteq_H$" ("+<") | "$\sqsupseteq_H$" (">+") | "$\cong_H$" ("=+=")

<*attr-op*> ::= "—" ("->") | "—" ("<-") | "=" ("=")

⑦ オブジェクト項

% 一般の <o-term> 用

<o-term> ::= <c-o-term>
| <*g-var*>
| <*exp-name*>
| <dot-term>
| <*self*>

<dot-term> ::= <o-term>"!"<lab>

% <o-term>, <m-id>, <lab>, <a-o-term> 用

<c-o-term> ::= <o-head> ["["<o-attr>-*list*(",")"]" ["|""{"<o-cnstr>-*list*(",")"}"]]
| <a-o-term>

<o-head> ::= <*basic-obj*>

<o-attr> ::= <lab> <*attr-op*> <o-term>
% 現在は<o-term>に<dot-term>は許さない
| <lab> <*attr-op*> <m-id>

<lab> ::= "1_"<c-o-term> % 現在は"1_"<*basic-obj*> のみ

<a-o-term> ::= <*l-var*>"@" <c-o-term>

<m-id> ::= "m_"<c-o-term>
| <*g-var*>
| <*self*>

% <o-term> の制約用

<cnstr-o-term> ::= <o-head> ["[" <o-attr>-*list*(",")"]"]
| <*l-var*>
| <*exp-name*>

<o-cnstr> ::= <*l-var*> <*sub-rel*> <cnstr-o-term>

<*self*> ::= "&self"

<*number*> ::= "0" | ··· | "9"

<*integer*> ::= <*number*>-*list*

<*upper-char*> ::= "A" | ··· | "Z"

<*lower-char*> ::= "a" | ··· | "z" | <*number*> | "_" | "&" | < 漢字 >

% <*lower-char*> の "&" は組込みオブジェクト項用に使用。

<*char*> ::= <*upper-char*> | <*lower-char*>

<*string*> ::= <*char*>-*list*

<*l-string*> ::= <*lower-char*> [<*string*>]

<*u-string*> ::= <*upper-char*> [<*string*>]

<*g-var*> ::= <*u-string*> | "_"<*string*>

<*l-var*> ::= <*u-string*> | "_"<*string*>

<*basic-obj*> ::= <*l-string*>

⑧ 問合せ

<query> ::= "?-" <query-body> [";" "&q_mode" "[" <q-mode>-*list*(",")"]"]
[";" <program-def>] "."
| "?-" "&open/" "[" <op-attr>-*list*(",")"]" "."
| "?-" "&close/" "[" <cl-attr>-*list*(",")"]" "."

<op-attr> ::= "&pgm_name=" <*pgm-name*>
| "&lib_name=" <*lib-name*>
| "&lib_name+=" "[" <*lib-name*>-*list*(",")"]"
| "&mode=" <*op-mode*>
| "&ver=" <*version*>

<cl-attr> ::= "&pgm_name=" <*pgm-name*>
| "&lib_name=" <*lib-name*>
| "&mode=" <*cl-mode*>
| "&exit=" <*cl-exit*>

<q-mode> ::= "&proc_mode=" <*proc-mode*>
| "&ans_mode=" <*ans-mode*>
| "&inheritance=" <*inheritance*>

<*op-mode*> ::= "&readonly" | "&exclusive"

<*cl-mode*> ::= "&end" | "&abort"

<*proc-mode*> ::= "&single" | "&multi"

<*ans-mode*> ::= "&normal" | "&minimal"

<*inheritance*> ::= "&all" | "&down" | "&up" | "&no" | [<*u-flag*>] <*integer*>

<cl-exit> ::= "&yes" | "&no"

<*version*> ::= <*string*>

【参考文献】

[成果報告 91-1] ICOT. 平成 2 年度電子計算機基礎技術開発成果報告書 ソフトウェア編. pp 415-447.

1991.

[成果報告 91-2] ICOT. 平成2年度発電設備診断システムの開発成果報告書 試作編(II). pp 403-430. 1991.

[横田 90] 横田、安川、高橋、西岡、平井、森田. "知識ベース・知識表現言語 *Quixote*". ICOT Technical Momo, 1990.

[Tanaka 91] H. Tanaka. *Protein Function Database as a Deductive and Object-Oriented Database. The Second International Conference on Database and Expert System Applications*, Berlin, Apr., 1991.