TM-1285

Quixote システム　第3版

津田　宏、横田　一正、森田　幸伯（沖）、
高橋　千恵（JIPDEC）、西岡　利博、
新部　裕（MRI）、小島　量、高田　葉子、
堀川　勉（管理研）、合田　光宏、
高瀬　真志（(株) アーティフィッシャルインテリジェンス）、
天沼　敏幸（MTC）、須田　智子（(株) ソフトウエアリサーチアソシエーツ）

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

Institute for New Generation Computer Technology

QUIXOTE システム第3版

津田宏　横田一正
(財) 新世代コンピュータ技術開発機構

森田幸伯
沖電気工業 (株)

高橋千恵
(財) 日本情報処理開発協会

西岡利博　新部裕
(株) 三菱総合研究所

小島量　高田葉子　堀川勉
(株) 管理工学研究所

合田光宏　高瀬真志
(株) アーティフィッシャルインテリジェンス

天沼敏幸
三菱電機東部コンピュータシステム (株)

須田智子
(株) ソフトウェアリサーチアソシエーツ

1　概要

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、知識表現機能を有する中核的な言語として知識表現言語 QUIXOTE の設計・開発を行っている [Yokota 92, Yasukawa 92]。

QUIXOTE はプログラム言語としては、演繹オブジェクト指向データベース (Deductive Object-Oriented Database: DOOD) 言語、もしくは包摂関係に基づく記号的な制約論理型 (Constraint Logic Programming: CLP) 言語と分類される。その特徴として以下のようなものが挙げられる。

- オブジェクトおよびオブジェクト識別性
  知識の基本的な表現単位はオブジェクトであり、それはオブジェクト識別子とプロパティ (メソッド) の集合とからなっている。オブジェクト識別子は拡張項として表現される。
- 包摂関係に基づく制約記述
  包摂関係 (汎化関係) はオブジェクト間の概念の関係であり、IS-A 関係に対応する。
- プロパティ継承
  オブジェクトはプロパティを持ち、その値は包摂関係に基づく制約によって与えられる。包摂関係にあるオブジェクト間ではプロパティは継承される。
- モジュール機能
  知識をモジュールによって分割・階層化し、それらの間に継承関係を導入することにより、効率的な知識記述を可能にする。
- データベース機能
  動的更新機能、永続性、入れ子トランザクションなどのデータベース機能をもっている。
- 仮説推論、仮説生成
  QUIXOTE オブジェクトの集まりとしてのデータベースは部分情報データベースと考えることができ、仮説推論と仮説生成 (アブダクション) によって、思考実験的な問合せを行うことができる。

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 / 仮説推論・生成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。

第3版処理系の設計・開発に関しては、

- 応用プログラムおよび各種インタフェースの拡充
- データベース更新機能の機能拡張
- 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、研究開発を行なった。

Quixote 第3版処理系は、UNIX および C 言語により記述されたクライアントと、KL1 で記述されたサーバから構成されており、それらは TCP/IP により通信する。

KL1 で書かれた *Quixote* 第3版サーバについては、第2版の相当部分を基本に以下のような機能拡張を行なった。

- 推論速度向上のため、内部データ構造・アルゴリズムを変更
- 第2版では不十分であった、データベース更新機能の機能拡張

従来第2版までは ESP および Pmacs により実現していたインタフェースは、第3版ではクライアントとして、C および X-Window、GNU Emacs により全面的に書き換えを行なった。これにより移植性とともに、(グラフィカル) ユーザインタフェースが向上した。具体的には、

- GNU-Emacs を拡張して、*Quixote* プログラムの実行を行なうことのできる Qmacs。
- QIF ライブラリ、QIF ユーティリティを用いて作成された端末からのユーザインタフェースである Qshell。
- X-Window を利用したグラフィカルユーザインタフェースによる知識オブジェクトの束構造、モジュールの階層構造、質問に対する解の導出過程などの表示。

により、構成されている。

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し、グラフィカルに結果を表示することが可能となった。

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、やりのこした研究課題として、以下のようなことが挙げられる。

- より大規模・広範囲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。
- プラットフォーム機能、プログラミング機能などに関する *Quixote* の言語仕様の拡張。
- 異種の知識ベースを統合するマルチ知識ベースとしての *Quixote*。
- 他言語・外部制約解消系とのインタフェース。データのやりとりに当たってのデータ型の導入。
- 応用システムの開発環境とのインタフェースの確立。

特に、これまでの *Quixote* の設計・開発では、並列性を推論の上でうまく生かすことはできなかった。並列推論アルゴリズムの改良のみならず、マルチエージェント指向言語のような考え方を用いて *Quixote* をとらえ直し、データ構造の段階から本質的な並列性を考えていく方向も考えられる。

以下、2節では、*Quixote* 第3版処理系の全体構成、3節では構文と内部データ構造について紹介する。4節では UNIX 上の C および X-Window、GNU-Emacs により作られた *Quixote* クライアントを、5節では KL1 で実装された *Quixote* サーバについて、それらの概要および仕様を説明する。6節は、*Quixote* 第3版の Qmacs という GNU-Emacs に基づいたインタフェースの使い方を記す。

2 全体構成

Quixote 第3版処理系は、UNIX 上のクライアントと PIMOS 上のサーバが TCP/IP を介して通信する構成になっている。図1はその全体構成図であり、図の左方がユーザインタフェースであるクライアント (*Quixote* client)、右方がサーバ (*Quixote* server) である。

クライアント (*Quixote* Client) は以下の構成になっている。

- Qmacs: ユーザインタフェース (1)。GNU-Emacs を利用した *Quixote* 処理系インタフェース。
- Qshell: ユーザインタフェース (2)。QIF Library を利用して端末からのインタフェースを提供する。
- QIF Library: Qshell のライブラリ集。項の変換 (parser/unparser)、.src データ構造の処理などを行なう。
- Window: 基本オブジェクトの束構造の表示、モジュールの階層構造の表示、解の導出過程の表示を行なう。

サーバ (*Quixote* Server) は以下の構成になっている。

- *Quixote* (TCP/IP) サーバ: KL1 側の TCP/IP サーバの低レベルモジュール
- KL1 IF: データ変換等の処理
- QS (*Quixote* (database) Server): 複数データベースの管理を行なう。

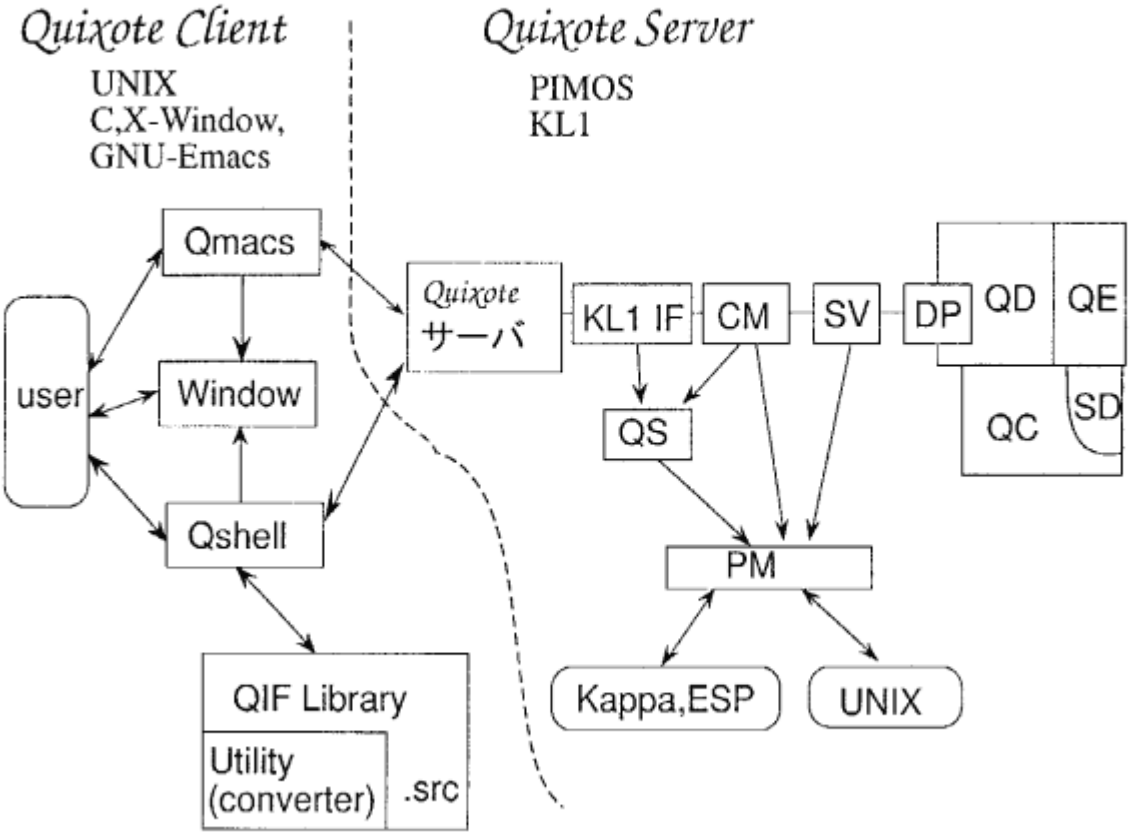

図 1: *Quixote* 第 3 版の全体構成図

- PM (Persistence Manager): データベース機能を実現する外部システムとのインタフェース。
- CM (Communication Manager): 項の変換処理の一部を行なう
- SV (SuperVisor): QD の各モジュールに命令を送る
- DP (DisPatcher): QD の各モジュールの管理を行なう
- QD (*Quixote* Data Manager): *Quixote* のオブジェクトの内部形式を保持する
- QE (*Quixote* Execution): *Quixote* の推論処理を行なう
- QC (*Quixote* Constraint handler): 制約解消系・単一化子など、項の基本操作のサブルーチンの集合
- SD (Solution Description manager): QC のモジュールの一つで、QE の解表現に対する処理のサブルーチンの集合

3 *Quixote* の構文・データ構造

本節では、*Quixote* 第 3 版の構文および内部データ構造について記す。

3.1 *Quixote* の構文 (Prolog I/F を含む)

1) 記法

```
[…]                 =   … は省略可能
<…>-list            =   <…> [<…>-list]
<…>-list("d")       =   <…> ["d" <…>-list("d")]
<…> ::= … {a|b} …   <=> <…> ::= … a … | … b …
```

2) 文字

```
<u-string>      ::= <upper-char> [<char-list>]
<l-string>      ::= <lower-char> [<char-list>]
<char-list>     ::= <char>-list
<char>          ::= <upper-char> | <lower-char>
<upper-char>    ::= "A" | .. | "Z"
<lower-char>    ::= "a" | .. | "z" | <number> | "_" | "&" | <漢字>
<integer>       ::= <number>-list
<number>        ::= "0" | .. | "9"
<string>        ::= """ <print-char>-list """
```

3) プログラム

```
<program-def>   ::= <b-pgm> ";;" <definition-list> "&end"
<program>       ::= <program-def> "."
<b-pgm>         ::= "&program" | "&pgm" | "&database" | "&db"        % 定数付加
<definition-list> ::= <definition>
                    | <definition-list> <definition>
<definition>    ::= <env-def> | <exp-def> | <obj-def> | <mod-def>
                    | <link-def> | <rule-def>
```

4) 環境

```
<env-def>       ::= <b-env> ";;" <env-list>
<b-env>         ::= "&environment" | "&env"
<env-list>      ::= <env>  |  <env-list> <env>
<env>           ::= "&name" "[" "&pgm-name" "="  <char-list> "]" ";;"
                    | "&author" "[" "&aut-name" "=" <char-list>"]" ";;"
                    | "&date" "[" "&date" "="       <char-list>"]" ";;"
                    | "&include" "[" <def-lib>-list(",") "]" ";;"
<def-lib>       ::= <lib-lab> "=" {<lib-name> | "{"<lib-name>-list(",")"}"}
<lib-name>      ::= <string>
<lib-lab>       ::= "&exp_lib" | "&pgm_lib" | "&sort_lib"
```

5) EXPRESSION

```
<exp-def>       ::= <b-exp> ";;" <exp-list>
<b-exp>         ::= "&expression" | "&exp"
<exp-list>      ::= <exp>  |  <exp-list> <exp>
<exp>           ::= <exp-name> "=" <exp-operation> ";;"
<exp-name>      ::= "e_"<char-list>
<exp-operation> ::= <o-term> | <mid>
  | "&del(" <b-obj>-list(",") ")" <exp-name>
  | "&add(" "[" <o-attr-list> "]" ["," "{" <o-cnstr-list> "}" ] ")" <exp-name>
  | "&abs(" <o-attr> ["," "{" <o-cnstr-list> "}" ] ")" <exp-name>
```

6) オブジェクト項間関係

```
<obj-def>       ::= <b-obj> ";;" <obj-sub-list>
<b-obj>         ::= "&object" | "&obj" | "&subsumption" | "&subsum"  % 定数付加
<obj-sub-list>  ::= <obj-sub> |  <obj-sub-list> <obj-sub>
<obj-sub>  ::= <b-obj> {"=<" | ">="} {<b-obj> | "{"<b-obj>-list(",")"}"}";;"
```

7) モジュール間関係

```
<mod-def>       ::= <b-mod> ";;" <m-sub-list>
<b-mod>         ::= "&module" | "&mod" | "&submodule" | "&submod"    % 定数付加
<m-sub-list>    ::= <m-sub>   |  <m-sub-list> <m-sub>
<m-sub>         ::= <mid> ">-" <m-desc> ";;"
<m-desc>        ::= <mid> | "(" <m-desc> ")"  | <m-desc> {"+" | "-"} <m-desc>
```

8) LINK

```
<link-def>      ::= <b-link> ";;" <link-list>
<b-link>        ::= "&link"
<link-list>     ::= <link>   |   <link-list> <link>
<link>          ::= "[" <link-name> ","
      ["{"<o-link>-list(",")"}"] "," ["{"<m-link>-list(",")"}"] "]" ";;"
<link-name>     ::= <l-string>
<o-link>     ::= "[" <o-term> "," {<o-term> | "{" <list-o-term-list> "}" } "]"
<m-link>        ::= "[" <mid> "," {<mid> | "{" <c-o-term>-list(",") "}" } "]"
```

9) ルール

```
<rule-def>      ::= <b-rule> ";;" <rules>
<b-rule>        ::= "&rule"
<rules>         ::= <rules-sub>  |  <rules> <rules-sub>
<rules-sub>     ::= <rule> ";;"  |  <mod> <rule> ";;"
                 |  <mod> "{" <rule>-list(";;") "}" ";;"
                 % TRIAL 用拡張
                 |  "&no_assume" <rule> ";;"
                 |  "&no_assume" <mod> <rule> ";;"
                 |  "&no_assume" <mod> "{" <rule>-list(";;") "}" ";;"
<mod>           ::= {<mid> | "{" <c-o-term>-list(",") "}" } "::"
<rule>          ::= ["(" <rule-lab> ")"] {<clause> | <u-clause>}
<rule-lab>      ::= <rule-id> ["," <inh-mode>]  | <inh-mode>
<rule-id>       ::= <char-list>
<inh-mode>      ::= "&l"["o"] | "&o"["l"]
<clause>        ::= <a-term> [{"<=" | ":-"} <body>]          % Prolog I/F に対応
<body>          ::= <props> ["|" "{" <a-cnstr-list> "}" ]
<props>         ::= <prop> ["," <props>]
<prop>          ::= [<m-lab>] <a-term>  | <o-term> <sub-rel> <o-term>
<m-lab>         ::= <mid> ":"
<u-clause>      ::= <a-term> {"<=" | ":-"} <u-body>          % Prolog I/F に対応
<u-body>        ::= <u-body> ";" <u-cluster>
                  | [<m-lab>] <a-term> ";" <u-cluster>
                  | {"+" | "-"} [<m-lab>] <a-term>
                  | <trans>  | <integrity>
<u-cluster>     ::=  [<m-lab>] <a-term>
                  | {"+" | "-"} [<m-lab>] <a-term>
                  | <trans> | <integrity>
<trans>         ::= "&begin_transaction" | "&bt"
                  | "&end_transaction"   | "&et"
                  | "&abort_transaction" | "&at"
<integrity>     ::= {"&consis" | "&inconsis"} "(" <i-check> ")"
<i-check>       ::= [<m-lab>] <a-term>  | "{" <a-cnstr-list> "}"
```

10) 属性項

```
<a-term>        ::= <o-term>
                |  <o-term> "/" "[" <a-attr-list> "]"
                |  <o-term> "/" "|" "{" <a-cnstr-list> "}"
                |  <o-term> "/" "[" <a-attr-list> "]" "|" "{" <a-cnstr-list> "}"
<a-attr-list>   ::= <a-attr>-list(",")
<a-attr>        ::= <a-lab>   <attr-op> <ind-term>
                  | <set-lab> <set-rel> <set-terms>
                  | <a-lab>   "->"      <set-terms>
                  | <set-lab> "<-"      <ind-term>
                  | "{"<a-lab>"}"   <attr-op> <ind-term>  % Query
                  | "{"<set-lab>"}" <attr-op> <ind-term>  % Query
<a-lab>         ::= <c-o-term>                    % 点けるラベルは限られている
                  | "&car" |  "&cdr"              % <list> 用組込みラベル
<sort>  ::= "s_"<o-head> "(" <o-term>-list(",") ")"
<ind-term>      ::= <o-term> | <sort>
<a-cnstr-list>  ::= <a-cnstr>-list(",")
<a-cnstr>       ::= [<m-lab>] <ind-term> <sub-rel> [<m-lab>] <ind-term>
                  | [<m-lab>] <set-lhs>  <set-rel> [<m-lab>] <set-terms>
                  | [<m-lab>] <ind-term> "&in"     [<m-lab>] <set-terms>
                  | [<m-lab>] <set-lhs>  "&ni"     [<m-lab>] <ind-term>
<set-lab>       ::= <b-obj>"*"
<set-terms>     ::= "{" <list-o-term-list> "}"  | <set-lhs>
<set-lhs>       ::= <o-term>"!"<set-lab>  | <set-var>
<set-var>       ::= <g-var>"*"
<attr-op>       ::= "->" | "<-" | "="
<sub-rel>       ::= "=<" | ">=" | "=="
<set-rel>       ::= "*<" | ">*" | "=*="
```

11) オブジェクト項

```
<o-term>        ::= <list-o-term>  | <g-var>  | <dot-term>
<list-o-term>   ::= <c-o-term>  | <exp-name>  | <list>
                  | <string> | <integer>
<c-o-term>      ::= <o-head>
                  |  <o-head> "[" <o-attr-list> "]"
                  |  <o-head> "[" <o-attr-list> "]" "{" <o-cnstr-list> "}"
                  |  <o-head> "(" <o-term>-list(",") ")"
                     % Prolog I/F に対応
                     % <o-head>"[$1="<o-term>",$2="<o-term>"," … "]" に変換
                  | <a-o-term>
<o-head>        ::= <b-obj>
<o-attr-list>   ::= <o-attr>-list(",")
<o-attr>        ::= <b-obj> "=" <o-term>
<o-cnstr-list>  ::= <o-cnstr>-list(",")
<o-cnstr>       ::= <l-var> "==" <cnstr-o-term>
<cnstr-o-term>  ::= <o-head> [ "[" <o-attr-list> "]" ]
                  | <l-var> | <exp-name>
<a-o-term>      ::= <l-var>"@"<c-o-term>
<list>          ::= "[]" | "["<o-term>-list(",")"]"
                  | "["<o-term>-list(",")""|"{<list>|<g-var>}]"
```

```
                % 論理的には list[car=<o-term>,cdr=<list>] の略記と考える
<mid>           ::= <c-o-term>  | <g-var>  | "&self"
<dot-term>      ::= <o-term>"!"<a-lab>
<g-var>         ::= <u-string> | "_"<char-list>
<l-var>         ::= <u-string> | "_"<char-list>
<b-obj>         ::= <l-string>
```

12) 問合せ

```
<query>         ::= "?-" <q-body> [<q-attach>] "."
                  | "?-" <u-body> [<q-attach>] "."
                  | "?-" (<a-term>) <q-body>  [<q-attach>] "."
                  | "?-" (<a-term>) <u-body>  [<q-attach>] "."
<q-body>        ::= <props> |  <props> "||" "{" <q-cnstr>-list(",") "}"
<q-cnstr>       ::= <mid> ">-" <mid>                        % TRIAL 用拡張
                  | <a-cnstr>
<q-attach>  ::= [";;" "&q_mode" "[" <q-mode-list> "]" ] [";;" <program-def>]
<q-mode-list>   ::= <q-mode>-list(",")
<q-mode>        ::= "&proc_mode"   "=" {"&single" | "&multi"}
                  | "&ans_mode"    "=" {"&normal" | "&minimal"}
                  | "&inheritance" "=" <inheritance>
                  | "&merge"       "=" {"&yes" | "&no"}
                  | "&explanation" "=" {"&on" | "&off"}
<inheritance>   ::= "&all" | "&down" | "&up" | "&no"
```

3.2 *QUIXOTE* 第 3 版プログラム記述例

2 つの例を示す。

1) サイダーに関する概念分類

```
&program;;
&subsumption;;        % 各種アトムおよび包摂関係の定義
    apple >= {macintosh,fuji,kokkou,indian_delicious};;
    apple =< {rose,food};;
&submodule;;          % モジュール関係の定義
    usa >- west;;
    uk >- west;;
&rule;;
japan::{cider/[source=soda_pop];;  % 日本では非アルコール飲料は無税
        drink[name=X]/[trade=no_tax] <= X/[alcohol=non]};;
west::cider/[source=apple, process=ferment];; % 西欧ではサイダーは
                                    % リンゴから蒸留により作られる
uk::cider/[alcohol=yes];;   % イギリスではサイダーはアルコール飲料
usa::cider/[alcohol=non];;  % アメリカではサイダーは非アルコール飲料
&end.
```

2) Bizet と Verdi に関する状況推論

```
&program;;
&subsumption;;          % アトムおよび包摂関係の定義
    nation >= {italy,france};;
&rule;;
```

```
hypothesis_a :: bizet/[nationality = italy] <=
    compatriots[per1=bizet,per2=verdi] ;;
        % A: 「ビゼーとヴェルディが同国人ならビゼーはイタリア人だ」
hypothesis_b :: verdi/[nationality = france] <=
    compatriots[per1=bizet,per2=verdi] ;;
        % B: 「ビゼーとヴェルディが同国人ならヴェルディはフランス人だ」
compatriots[per1=X,per2=Y] <=          % 同国人の定義
    X/[nationality=N1], Y/[nationality=N2] ||
    {N1=<nation,N2=<nation,Ni==N2} ;;
bizet;;                    % bizet および verdi オブジェクトが存在する
verdi;;
&end.
```

3.3 *QUIXOTE* の内部データ構造

ここでは、*QUIXOTE* の代表的なデータ構造の内部データ構造を記す。

1) 記法

```
(AAA | BBB)-list        AAA または BBB を要素とするリスト
(AAA | BBB)-vector      AAA または BBB を要素とするベクタ
```

2) オブジェクト項・ドット項

```
<Q項>    ::= <変数> | <bobj> | <type> | <set> | <list> | <string> | <PrologTerm>
<ID>     ::= 自然数
<アトム> ::= {KL1アトム,<ID>}        % <ID>は基底オブジェクト項ID
<変数>   ::= 自然数
<bobj>   ::= <アトム> | <数値>
<数値>   ::= {Num}                   % Numは自然数
<ラベル> ::= <アトム> | <数値>        % 数値は基底(複合)オブジェクト項ID
<type>   ::= {<Head>,<Add>,<L1>,<V1>,...,<Ln>,<Vn>}
        <Head> ::= <アトム>          % オブジェクト項ヘッド
        <Add> ::= <ID> |             % 基底オブジェクト項ID
                  0                  % パラメトリック項の場合
        <Li> ::= <ラベル> % ラベル(IDの昇順にソートされている)
        <Vi> ::= <Q項>               % 値
<set>    ::= {1,<Elements>}
        <Elements> ::= <リスト>              % 基底オブジェクト項のリスト
                % 基底オブジェクト項ID比較による昇順にソートされている
<list>   ::= {2,<Elements>}
        <Elements> ::= <リスト>              % <Q項>のリスト
<string> ::= {3,<String>}
        <String> ::= kl1のストリング(要素は16ビット幅)
<PrologTerm> ::= {4,{Functor,Arg1,...,Argn}}          % Prologの項
        <Functor> ::= <アトム>
        <Argi> ::= <Q項>
<dot>    ::= {dot,<Add>,<Module>,<DHead>,<Label>}
        <Add> ::= <ID>|              % 基底ドット項ID
                  0                  % パラメトリックドット項の場合
        <Module> ::= <Q項>           % モジュール
        <DHead> ::= <Q項> % ドット項ヘッド
        <Label> ::= <ラベル>         % ラベル
```

3) 変数環境

```
<変数環境> ::= {<変数環境ベクタ>,<ドット項参照ベクタ>}
<変数環境ベクタ> ::= <変数制約>-vector
                    % 第0要素には使用済み領域を表す数値が入る
<変数制約> ::=    0 | {<Smaller>,<Larger>} |                % 通常変数
                  {<Smaller>,<Element_of>,<Larger>}|
                  [] | [<Smaller>,<Larger>] |               % 集合変数
                  <Q項> |
<Smaller> ::= <Q項>-list
<Larger> ::= <Q項>-list
<Element_of> ::= <Q項>-list | <代入式>-list
<代入式> ::= {<変数>,:=,式}
<ドット項参照ベクタ> ::=  ( 0 | <dot> | <dot>-list )-vector
```

ドット項テーブルの逆引き（ドット項対応変数 -> ドット項）を行なう。
対応するドット項が複数ある場合には、それらのリストを値とする。

4) 変数環境ベクタの例

a=<Y, X=<Y, Z+ +< {a,b} のときの変数環境ベクタは次のようになる。

```
    0    1(X)      2(Y)        3(Z)
{3,{[],[2]},{[{a,5},1],[]},[[],[{1,[{a,5},{b,6}]}]],...}
```

5) ルールのデータ構造

```
<rule> ::= {<rule-class>, <rule-id>, <module>, <head>, <head-dot-constraint>,
            <body-list>,
            <number-of-vars>, <rule-varenv>,              % QDでのみ使用
            <assumption-control>, <trace-control>}
<rule-class>     ::= '&update' | '&noupdate'
<rule-id>        ::= ルール識別子
<module>         ::= <アトム> | <type> | <変数>
<head>           ::= <アトム> | <type> | <変数>
<head-dot-constraint> ::= <ドット項テーブル>
<body-list>      ::= <body-cluster>-list
<number-of-vars> ::= 数値                  % ルール中の変数の数
<rule-varenv> ::= <変数環境>               % ルールの変数環境
<assumption-control> ::= on | off          % 仮説制限の制御フラグ
<trace-control>      ::= on | off          % トレースの制御フラグ
<body-cluster> ::= <noupdate-body-cluster> | <update-body-cluster>
<noupdate-body-cluster> ::= {<subgoals>, <body-dot-constraint>}
<subgoals> ::= <subgoal>-list
<subgoal>  ::= {<module>, <object>, <labels>, <optimization>}
<labels> ::= <ラベル>-list
<optimization> ::= 0
<body-dot-constraint> ::= <ドット項テーブル>
<update-body-cluster> ::= <transaction-controller> | <update-controller>
                        | <ordinary-cluster>
<transaction-controller> ::= <begin-transaction> | <end-transaction>
        | <abort-transaction> | <consis> | <inconsis>
<begin-transaction> ::= '&bt'
<end-transaction>   ::= '&et'
<abort-transaction> ::= '&at'
```

```
<consis> ::= {'&consis',<consis-subgoal>, <cluster-dot-constraint>,
              <check-vars>,<variable-env>}
<inconsis> ::= {'&inconsis',<consis-subgoal>, <cluster-dot-constraint>,
              <check-vars>,<variable-env>}
<consis-subgoal> ::= <subgoal> | '$void'
<cluster-dot-constraint> ::= <ドット項テーブル>
<check-vars> ::= <変数>-list
<variable-env> ::= <変数環境>
<update-controller> ::= <assert-object> | <assert-attribute>
        | <retract-object> | <retract-attribute>
<assert-object> ::= {'+obj',<subgoal>,<variable-env>}
<assert-attribute> ::=
        {'+dot',<subgoal>,<dot-constr-exprs>,<variable-env>}
        | {'+par',<subgoal>,<dot-constr-exprs>,<variable-env>}
<dot-constr-exprs> ::= <constr-expr>-list
<constr-expr> ::= {<dot>,<dot-var>,<rel>,<term>}
<dot-var> ::= <変数>
<term> ::= <変数> | <type> | <bobj>
<retract-object> ::= {'-obj',<subgoal>,<variable-env>}
<retract-attribute> ::= {'-dot',<dot>,<variable-env>}
<ordinary-cluster> ::=
        {<subgoals>, <cluster-dot-constraint>, <check-vars>, <variable-env>}
```

6) その他のデータ構造

```
<単一化子> ::= ( {<変数>,<Q 項>} | {<変数>,<変数制約>,<Q 項>} )-list
<変数対応ベクタ> ::= ( 0 | <Q 項> )-vector
```

4 *Quixote* クライアント

4.1 *Quixote* クライアントの構成

本節では *Quixote* クライアントの構成について述べる。

Quixote クライアントは *Quixote* サーバを利用する環境であり、UNIX マシンから *Quixote* を利用することを目的としている。*Quixote* クライアントには、Emacs から *Quixote* を利用する Qmacs と、端末から *Quixote* を利用する Qshell の二つがある。また、束、モジュール関係をグラフィカルに表示する X11 のインタフェースもある。

Quixote クライアントは *Quixote* サーバと TCP/IP を用いて通信する。そこで用いる通信プロトコルを QIF プロトコルと呼ぶ。QIF プロトコルを実装し、*Quixote* を利用する関数を C のライブラリとして提供するのが QIF ライブラリである。

Quixote サーバと *Quixote* クライアントの全体の構成は図 2 のようになる。それは QIF プロトコル、API、ユーザインタフェースの三つのインタフェースで切られる。

- QIF プロトコル
  Quixote サーバと Unix 側クライアントとのプロトコルである。行を単位とするプロトコルで、コマンド実行要求、実行結果は KL1 のタームで表現される。
- API
  Quixote サーバを利用するための関数インタフェース。C 言語で呼び出す関数と用いるデータ構造の定義を与える。
- ユーザインタフェース
  ユーザが利用するインタフェース。ユーザインタフェースには三種類ある。すなわち、端末から使う Qshell、Emacs から使う Qmacs、および X11 のグラフィカルユーザインタフェースである。

Quixote クライアントの構成要素を以下に示す。

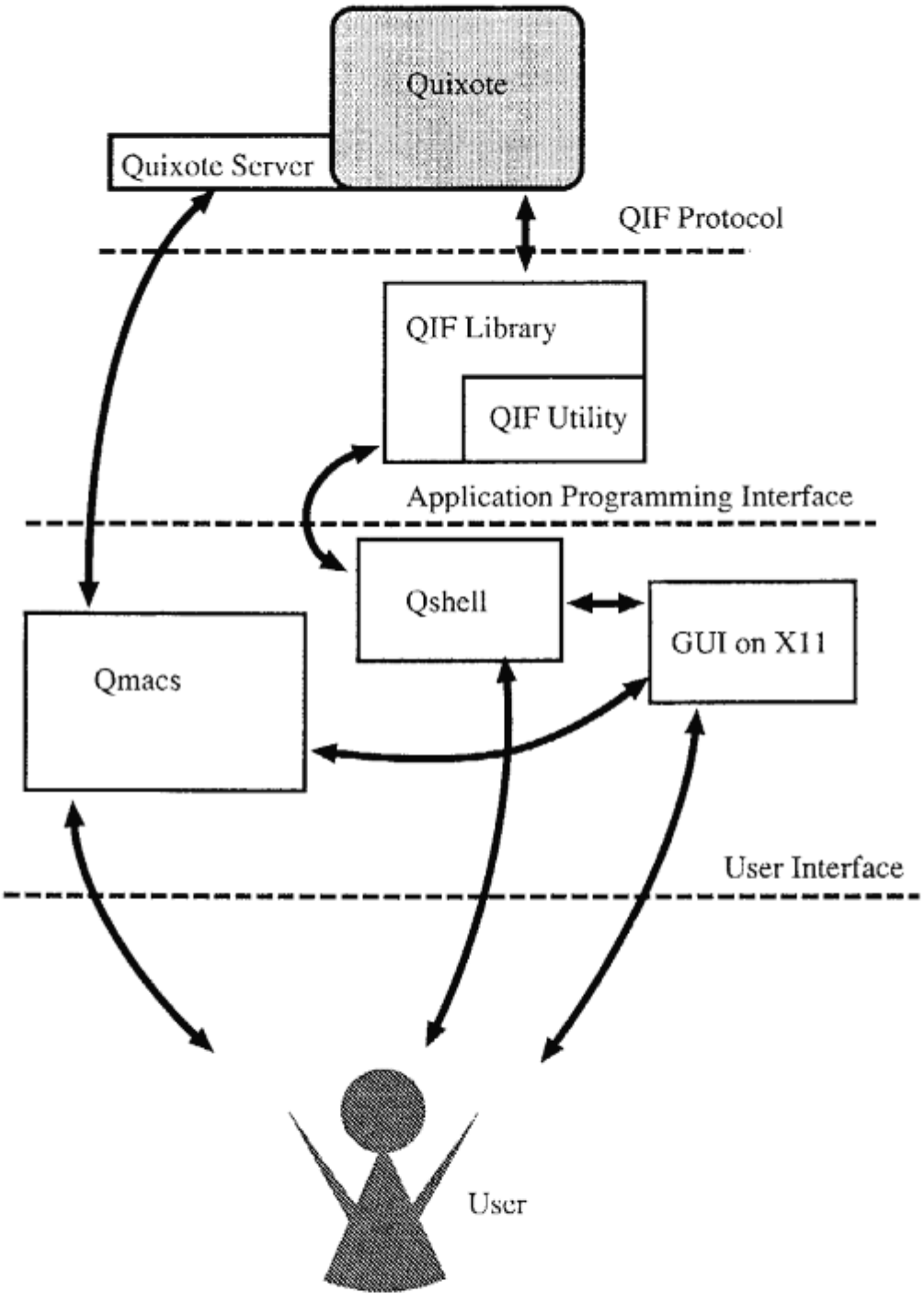

図 2: QUIXOTE クライアントの構成

- QIF ライブラリ
  QIF プロトコルに関係する通信の部分を実装する部分と、パーサ、アンパーサ、プリティプリンタなどのユーティリティの部分からなるライブラリ。
- Qmacs
  Emacs から *Quixote* を使うユーザインタフェース。*Quixote* サーバと直接通信する。GUI on X11 の起動もサポートする。
- Qshell
  QIF ライブラリ、QIF ユーティリティを用いて作成された端末からのユーザインターフェースである。インタラクティブなコマンド実行、記述ファイルによるバッチコマンド実行の両方の機能を提供する。インタラクティブなコマンド実行の場合、行編集、コマンド履歴、ヘルプ機能を提供する。GUI on X11 の起動もサポートする。
- X11 グラフィカルユーザインタフェース
  X11 上で実現するグラフィカルユーザインタフェース。Athena Widget を用いている。

4.2 QIF ライブラリ

Quixote クライアントプログラムより呼ぶことのできる C の関数ライブラリを作成した。このライブラリは以下の関数の集まりである。

parse:	*Quixote* ソースプログラムを .src と呼んでいる内部構造に変換する
unparse:	.src を整形 (プリティプリント) した *Quixote* ソースプログラムに変換する
conv:	.src をターム形式の文字列に変換する
rev:	ターム形式の文字列を.src に変換する
.src アクセス関数:	.src に read/write/update などのアクセスをする

また、これらを組合せて、

qxt2term:	*Quixote* ソースプログラムをターム形式の文字列に変換するプログラム
term2qxt:	ターム形式の文字列を *Quixote* ソースプログラムに変換するプログラム
pp:	*Quixote* ソースプログラムを整形（プリティプリント）するプログラム

なども作成した。

以下それぞれの関数を説明する。

- parser
  以下の形式の関数で、*Quixote* ソースプログラム、質問を .src に変換する。核の部分は yacc,lex で作成した。.src のアクセス関数を使っている。

```
PseudoObject *parse(FILE *stream)
    stream: QUIXOTE ソースプログラム
    関数値: .src
```

  構文エラーの場合、

```
syntax error: line xx\n
```

  を stderr に出力する。xx はエラーの行またはエラーの行の次を示す行番号である。

- unparser
  以下の形式の関数で、.src を ターム形式の文字列 に変換する。.src の アクセス 関数を使っている。

```
unparse(PseudoObject *object, FILE *stream)
    object: .src
    stream: ターム形式文字列
```

- conv
  以下の形式の関数で、ターム形式の文字列を .src に変換する。核の部分は yacc,lex で作成した。.src のアクセス関数を使っている。

```
PseudoObject *conv(FILE *stream)
    stream: ターム形式文字列
    関数値: .src
```

- rev

以下の形式の関数で、.src を *Quixote* ソースプログラム、答えに変換する。.src の アクセス 関数を使っている。

```
rev(PseudoObject *object, FILE *stream, int n)
    object: .src
    stream: quixote プログラム
    n:      pretty-print の形式。
            (例) n=1: ";;" で改行する。xxx-def の中の "&xxx;;"
                      以外の行は indent される。
```

- qxt2term

Quixote ソースプログラムをターム形式の文字列に変換するプログラム。以下の形式で起動すると quixote-program-file にある *Quixote* ソースプログラムを、ターム形式の文字列に変換し stdout に出力する。

```
qxt2term quixote-program-file または、qxt2term <quixote-program-file
```

- term2qxt

ターム形式の文字列を *Quixote* ソースプログラムに変換するプログラム。以下の形式で起動すると quixote-term-file にあるターム形式文字列を *Quixote* ソースプログラム に変換し stdout に出力する。

```
term2qxt\ quixote-term-file または、term2qxt\ <quixote-term-file
```

- pp

Quixote ソースプログラムを整形 (ソリティンリント) するプログラム。以下の形式で起動すると quixote-program-file にある *Quixote* ソースプログラムを、整形し stdout に出力する。

```
pp quixote_program_file または、pp <quixote_program_file
```

- .src

クライアント側の .src データは、KL1 の .src データを C 言語の構造体または共用体で表現したものである。.src データはアクセス関数によりクライアントプログラムからアクセスすることができる。

以下に $Program および $O_term のデータを例として示す。$Program データは C 言語データでの表現が構造体のデータであり、$O_Term データは共用体のデータである。

$Program データ

$Program の KL1 でのデータを以下に示す。

```
$Program = program($Env_def,$Exp_def,$Obj_def,$Mod_def,$Link_def,$Rule_def)
```

以下は、上記の $Program のデータを C 言語の構造体の形で表したものである。

```
struct  program{
        TypeDescripter  tag;
        EnvDef          *env_def;
        ExpDef          *exp_def;
        ObjDef          *obj_def;
        ModDef          *mod_def;
        LinkDef         *link_def;
        RuleDef         *rule_def;
                        };
```

構造体のメンバの tag はデータの識別子である。他のメンバは、それぞれのデータへのポインタである。

$O_term データ

$O_term の KL1 でのデータを以下に示す。

```
$O_term = o_term(c_o_term,$C_o_term)
        | o_term(var,$Var)
        | o_term(dot,$Dot)
        | o_term(list,$List)
        | o_term(non_struct,$Non_struct)
```

以下は、上記の $O_term のデータを C 言語の共用体の形で表したものである。

```
union o_term{
        Typedescripter      tag;
        Prolog              prolog;
        COTerm              c_o_term;
        Var                 var;
        Dot                 dot;
        List                list;
        NonStruct           non_struct;
        };
```

アクセス関数には、次のようなものが用意されている。これらのアクセス関数を利用しクライアントプログラムから、.src データにアクセスする。

- 引数のデータが、指定した種類のデータであるかどうか調べる。
- 構造体データの全メンバの値を得る。
- 構造体データの指定したメンバの値を得る。
- 構造体または共用体のデータを新しく作り、各メンバを引数で指定した値で初期化する。
  引数で示される構造体または共用体のデータをコピーする。
- 構造体データの各メンバを引数で指定した値にする。
- 構造体データのひとつのメンバを引数で指定した値にする。
- 構造体または共用体のデータを新しくつくり、各メンバを NULL で初期化する。
  構造体または共用体のデータを free する。
  構造体データのリスト構造になったメンバのリストの先頭に、引数で示されたデータをつなぐ
- 構造体データのリスト構造になったメンバのリストの最後に、示されたデータをつなぐ

4.3 Qmacs

- Qmacs の概要

  Qmacs は UNIX 上の *Quixote* クライアントであり、エディタ型のユーザインタフェースを提供する。Qmacs は GNU Emacs に *Quixote* コマンドを追加したものである。GNU Emacs の Emacs Lisp を用いて、機能拡張を行なった。

  Qmacs は以下のモジュールで構成している。

comm.el	サーバとの通信
result.el	サーバからの結果の解析
lex.el	*Quixote* の字句解析
parse.el	*Quixote* の構文解析
server-s.el	サーバセッションコマンドの処理
databases.el	データベースセッションコマンドの処理
trace-s.el	トレース / インスペクトセッションコマンドの処理
qshell.el	コマンドシェル機能
qxt-mode.el	Qxt モード編集機能

- サーバとの通信

  サーバとの通信における主要関数の説明を行なう。

qxt-start(host)　　*Quixote* の起動を行なう関数である。
open-network-stream を用いて、host のサーバと接続する。

qxt-filter (process string)　サーバからの送信を受信する関数である。
サーバが処理結果をクライアントに送ると、この関数が起動する。

qxt-send-command (line)　サーバに処理を依頼する関数である。サーバへコマンドを送信する。

- サーバからの結果の解析

qxt-term2lisp (string)　サーバから送られてきた文字列を Lisp で処理するために、
Lisp のオブジェクトに変換する関数である。

qxt-result(String)　サーバから送られてきた文字列を解析して、
対応する処理関数にディスパッチする関数である。

qxt-status-error(Status)　サーバから送られてきたステータスパラメータを解析して、エラー処理を行なう。

- *Quixote* の字句解析

get-next-char ()　一文字を先読みする関数である。

qxt-lex ()　字句解析を行なう関数である。

- *Quixote* の構文解析

advance (&optional token-expected)　トークンを一つ先読みする関数である。

qxt-syntax-error (&optional string)　文法エラーが発生した場合に処理を行なう。

program ()　プログラムのパージングを行なうエントリ関数である。

query ()　質問のパージングを行なうエントリ関数である。

a-term (&optional o-term_preread)　属性項のパージングを行なうエントリ関数である。

o-term ()　オブジェクト項のパージングを行なうエントリ関数である。

- サーバセッションコマンドの処理

qxt-quit (Yes)　*Quixote* を終了するコマンドである。

qxt-check-status (Session)　コマンド送信時に Qmacs のセッションチェック、
通信状態チェックを行なう関数である。
コマンド実行時に各関数から呼ばれる。

qxt-create-database (start end database-name)
データベースを生成するコマンドを送信する関数である。

qxt-result-create-database(Result)　データベースを生成するコマンドの実行結果を
受信した後の処理を行なう関数である。

qxt-qxt2src-D (text)　*Quixote* パーサを起動するエントリ関数である。
qxt-create-database から呼ばれる。
Emacs lisp で構文解析を行なう。

qxt-qxt2src-I (text)　*Quixote* パーサを起動するエントリ関数である。
qxt-create-database から呼ばれる。
C で書かれたパーサを起動し、構文解析を行なう。

qxt-open-database(DatabaseName Release DirectoryNames OpenMode FileType)
データベースをオープンするコマンドを送信する関数である。

qxt-result-open-database (Result)　データベースをオープンするコマンドの実行結果
を受信後の処理を行なう関数である。

- データベースセッションコマンドの処理

qxt-query (start end)	データベースに対する質問を送信する関数である。
qxt-result-query(Result)	質問の実行結果を受信後の処理を行なう関数である。
qxt-display-answer(AnswerElement)	質問の答を表示形式に変換し、バッファ上に表示する関数である。
qxt-close-database (End)	データベースをクローズするコマンドを送信する関数である。
qxt-delete-database()	データベースを削除するコマンドを送信する関数である。
qxt-show-lattice()	束のデータを要求するコマンドを送信する関数である。
qxt-result-show-lattice()	サーバから送信された束データを表示する関数である。
qxt-lisp2qxt(obj)	Lips オブジェクトをユーザが見える *QUIXOTE* の文字列に変換する関数である。
qxt-oterm2src (text)	入力されたコマンド中のオブジェクト項を サーバが受けとる.src 形式の文字列に変換する関数である。

- トレースセッションコマンドとインスペクトセッションコマンドの処理

qxt-set-trace-mode(TraceMode)	トレースモードをサーバに送信する関数である。
qxt-set-gate(GateID Switch)	トレース用のゲートをサーバに送信する関数である。
qxt-spy-at-rules(Rules)	スパイ設定をサーバに送信する関数である。
qxt-unspy-at-spypoints(SpyPoints)	スパイ設定の解除をサーバに送信する関数である。
qxt-trace-event(Result)	サーバから送られてきたトレース情報を表示する関数である。
qxt-execute-step()	トレースのステップ実行を送信する関数である。
qxt-execute-spy()	トレースのスパイ実行を送信する関数である。
qxt-inspect()	サーバにインスペクトセッションを送信する関数である。
qxt-inspect-assumption(SolutionNumber)	サーバにゴールの仮定部のデータ要求する関数である。
qxt-inspect-conclusion(SolutionNumber)	サーバにゴールの結論部のデータ要求する関数である。
qxt-inspect-variable(SolutionNumber VariableName)	サーバにゴールの変数データ要求する関数である。
qxt-quit-inspect()	インスペクトセッションの終了を送信する関数である。

- コマンドシェル機能

qxt-shell ()	バッファ上の *QUIXOTE* コマンドを一括実行する関数である。
qxt-wait ()	コマンドをサーバに送る時、 その前に送った *QUIXOTE* コマンドが終了するまで、待つ関数である。

- Qxt モード編集機能

qxt-mode ()	.qxt ファイルを編集する時、Qxt モードを設定する関数である。

4.4 Qshell

Qshell は *QUIXOTE* サーバを利用するクライアントプログラムで、以下の二つの機能を提供する。

- tty 端末から *QUIXOTE* を使うユーザインタフェース (インタラクティブ実行)
- ファイルに記述した処理を *QUIXOTE* サーバに要求し、処理する機能 (バッチ実行)

4.4.1 インタラクティブ実行の時に使える機能

インタラクティブモードでは行編集、コマンド履歴、およびコマンド説明 (ヘルプ) の各機能を利用することが出来る。

- 行編集機能
  通常の文字入力の他に、カーソル移動、挿入、削除などの機能を提供する。これにより、入力における誤りをカーソルを移動させて訂正することが出来る。デフォルトのキーバインディングは Emacs のものである。これはユーザ毎にカスタマイズの設定が可能である。
- コマンド履歴機能
  Qshell では各コマンドの起動の情報が保存される。この履歴を利用し、コマンドを再実行することが出来る。行編集機能と合わせて使うことにより、あるコマンドを引数でいろいろに変化させて実行するなどの処理が容易に行なえる。

- ヘルプ機能
  コマンドの使い方を Qshell 内で参照する機能である。ヘルプ機能には、あるセッションにおいて使えるコマンドの一覧表示 (コマンドの簡単な説明を含む) と、詳細な説明表示がある。

4.4.2 セッション

Qshell には *Quixote* サーバの状態に合わせて、以下のように四つのセッションがある。

- サーバセッション
- データベースセッション
- トレースセッション
- インスペクトセッション

4.4.3 Qshell の構成

Qshell はシェルである Qshell 本体と各セッションのコマンド群から構成される。

Qshell コマンド群の各ファイルは以下に示す表 1のように、あるディレクトリ (Qshell の起動時に指定可能、ここでは $QSHELL とする) に置かれる。

表 1: Qshell のセッションとコマンド群のディレクトリ構成

ディレクトリ	区分
$QSHELL/ss	サーバセッションコマンド
$QSHELL/db	データベースセッションコマンド
$QSHELL/tr	トレースセッションコマンド
$QSHELL/in	インスペクトセッションコマンド

4.4.4 Qshell の起動

Qshell の起動の形式は以下の通りである。

```
% qshell [ -d qshell-directory ] [ -f config-file-name ] [ server ]
```

ここで、*qshell-directory* は Qshell コマンド群があるディレクトリを、*config-file-name* はコマンド群の設定ファイルのファイル名、および *server* は *Quixote* の動いているサーバのホスト名である。

qshell-directory を指定しない場合、`/usr/local/qshell/` が用いられる。*config-file-name* を指定しない場合、`qsh.conf` が用いられる。*server* を省略した場合、環境変数の `QXTSERVER` が参照される。環境変数の `QXTSERVER` が設定されておらず、*server* を省略した場合は、エラーとして Qshell は終了する。

Qshell の中では以下のようにプロンプトで現在のセッションを表示する。

```
ss%     サーバセッション
db%     データベースセッション
ts%     トレースセッション
is%     インスペクトセッション
```

4.4.5 Qshell のコマンド文

Qshell においてユーザはコマンド文を入力し、コマンドを実行する。ここで、コマンドには Qshell メタコマンドと Qshell コマンドがある。Qshell は一行の入力をコマンド文として解析し、適切なコマンドを起動する。

コマンド文はコマンド名、コマンドに対する引数、およびリダイレクションからなる。リダイレクションにより、コマンドに与える入力をファイルから読み込んだり、コマンドの実行結果をファイルに保存することが出来る。

以下にコマンド文の文法を示す。

- 字句の定義

```
WORD                [a-zA-Z0-9_-./,+@:]+ あるいは \".*\"
GREATER_GREATER     >>
```

- 構文の定義

```
line: word_list '\n'
  | word_list redirections '\n'
  | '\n'
  | error '\n'
  ;
word_list: WORD
  | word_list WORD
  ;
redirection: '>' WORD
  | '<' WORD
  | GREATER_GREATER WORD
  ;
redirections: redirection
  | redirections redirection
  ;
```

4.4.6 Qshell メタコマンド

Qshell メタコマンドとは *Quixote* サーバとの通信を行なわず、クライアント側でローカルに処理するコマンドのことである。このコマンドは、四つのセッションのどのセッションでも利用することが出来る。

Qshell のメタコマンドには以下のものがある。

quit	*Quixote* と接続を切り、Qshell を終了する。
help	現在の session で使えるコマンド一覧を表示する。 引数をもえた時はそのコマンドについて説明を表示する。
shell	UNIX コマンドを実行する。
lcd	UNIX 側のカレントディレクトリを変更する。
take	ファイルからコマンド文を読み、実行する。

4.4.7 Qshell 本体と Qshell コマンドとのインタフェース

Qshell 本体は、Qshell コマンドを一般の shell から渡されるのと同じように、Qshell コマンドを呼び出す。Qshll コマンドを C で記述する場合、通常のように関数 main () が引数 argc、argv が設定されて呼ばれると考えて良い。一般の shell と異なる点は、標準入出力 (stdin. stdout. stderr) の他に、*Quixote* サーバとの通信のために、ネットワーク入出力 (netin. netout) が開いているということである。(この netin、netout は QIF ライブラリによって利用できる。)

Qshell コマンドの実行は以下のような手順をとる。

1) 引数と stdin から受けたユーザからの入力を解析する
2) netout に出力し、qxt サーバに要求を送る
3) netin を読み出し、*Quixote* サーバから結果を受け、解析する
4) 結果を適切に加工し、stdout に出力を行なう

Qshell コマンドは関数 exit () の返り値によって、セッションの遷移を Qshell 本体に伝える。表 2に返り値とセッションの遷移を示す。

4.4.8 コマンド設定ファイル

各コマンド群の directory にコマンド設定ファイルを用意する必要がある。このファイルの名前は Qshell の起動時に指定でき、デフォルトは `qsh.conf` である。

このファイルの内容は複数のエントリからなる。エントリ間は、空行で区切られる。

エントリは以下の書式に従う。

```
command-name  command-path      explanation
```

表 2: 返り値とセッションの遷移

値	状態
0	そのまま
1	サーバセッション
2	データベースセッション
3	トレースセッション
4	インスペクトセッション

ここで command-name は Qshell のコマンド名、command-path はそのコマンドを実行するコマンドのファイル名、explanation はコマンドの説明である。

コマンドの説明は文字列で複数行にまたがる。これは、最初の一行は簡潔な説明、二行目からは詳細な説明と二種類の説明を含む文字列である。簡潔な説明は help コマンドにおいて、そのセッションで使えるコマンドの一覧に表示するときに用いられる。詳細な説明は、そのコマンドを指定して help コマンドを利用した時の表示に用いられる。

コマンド設定ファイルの例を以下に示す。

```
create-database       crd     Create Database
Create database Named NAME.  Second argument is optional.
Second argument is FILENAME which include Quixote program.

cd cd         Change Directory
Change directory to DIR.  Notice that DIR is directory on Quixote
server.  If you want to change current directory of Qshell itself,
use lcd (local cd) command.

ls ls         List Directory
List directory of DIR.  Notice that DIR is directory on Quixote
server.  If you want to list directory of Qshell side, use
shell command to invoke the Unix command ls (% shell ls).

mkdir mkdir Make Directory
Make directory named DIR in Quixote server.

pwd pwd Print Working Directory
Print Working Directory of Quixote server.
```

4.5 Window 関係

Quixote 第二版ではグラフィカルユーザインタフェースを UNIX および X11 Window System 上で動かすことが目的であり、Lattice Window は X11R5 の国際語対応 Athna Widget および X11 Toolkit を使って作成されている。

4.5.1 Lattice Window

Lattice Window は *Quixote* データベース中のオブジェクトの包摂関係を束構造としてグラフィカルに表示するウィンドウで、Qmacs または Qshell から起動される。Lattice Window は起動後、Qmacs または Qshell から送られる show_lattice の実行結果である KL1 のターム文字列からオブジェクトの上下関係を解析し、ウィンドウ上に束構造を表示する (図 3)。図の表示位置はスクロールバーを使って変更することができる。また「検索」ボタンを使って束構造で表示したオブジェクト名を検索することができる。「検索」ボタンをクリックすると、オブジェクト名を文字コード順に表示したダイアログ (図 3) が表示され、ここで検索したいオブジェクトを選択すると束構造内の該当するオブジェクトが反転表示される。検索を終了する場合には検索ダイアログの「閉」をクリックするか、メインウィンドウの「検索」を再クリックする。Lattice Window を終了する時は「終了」ボタンをクリックする。

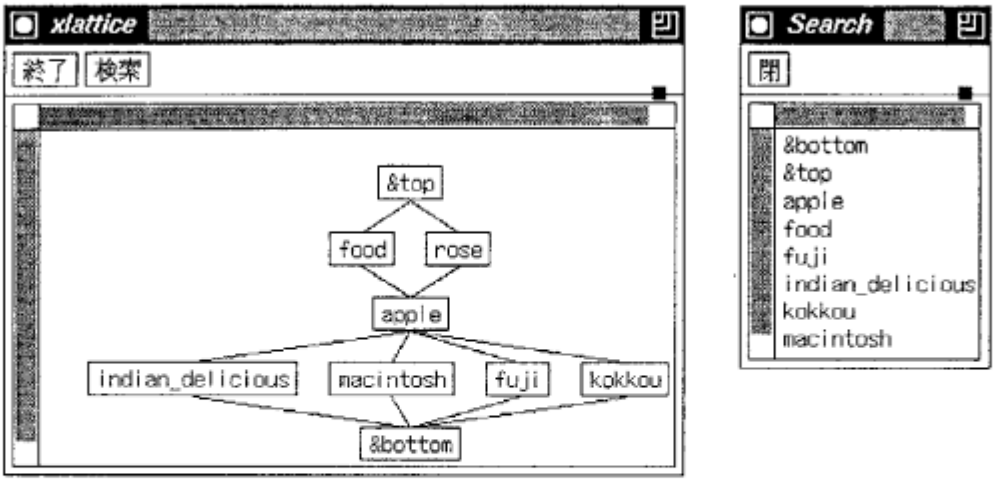

図 3: Lattice Window

4.5.2 Module Hierarchy Window

Module Hierarchy Window は *Quixots* データベース中のモジュールの関係を木構造としてグラフィカルに表示するウィンドウで、Qmacs または Qshell から起動される。

Module Hierarchy Window は起動後、Qmacs または Qshell から送られる show_module_hierarchy の実行結果からモジュールの上下関係およびルールを解析し、メインウィンドウにモジュール関係を木構造で表示する (図 4)。図の表示位置は

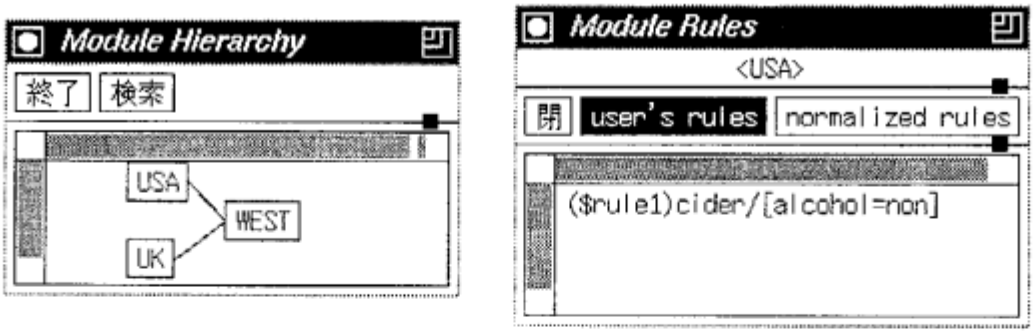

図 4: Module Hierarchy Window

スクロールバーを使って変更することができる。Lattice Window と同様に Module Hierarchy Window でも「検索」ボタンを使って木構造で表示したモジュール名の検索を行なうことができる。(図 4)。また、Module Hierarchy Window ではメインウィンドウに表示された木構造の各ノードがボタンになっていて、クリックするとそのモジュールに定義されたルールを表示するためのメニューが現れ、そこで選択したルールがルールダイアログに表示される (図 4)。ルールダイアログでは User's Rules/Normalized Rules と書かれたボタンがトグルボタンとなっており、表示中のルールを表すボタンは反転表示される。また、もう一方のボタンをクリックすることによって表示するルールを切替えることができる。Module Hierarchy Window を終了する時は「終了」ボタンをクリックする。

4.5.3 Answer Window

Answer Window は 解の導出木（説明データ）をグラフィカルに表示するウィンドウで、Qmacs または Qshell から起動される。起動後、Qmacs または Qshell から送られる解の導出木を解析し、ウィンドウ上に木構造を表示する (図 5)。導

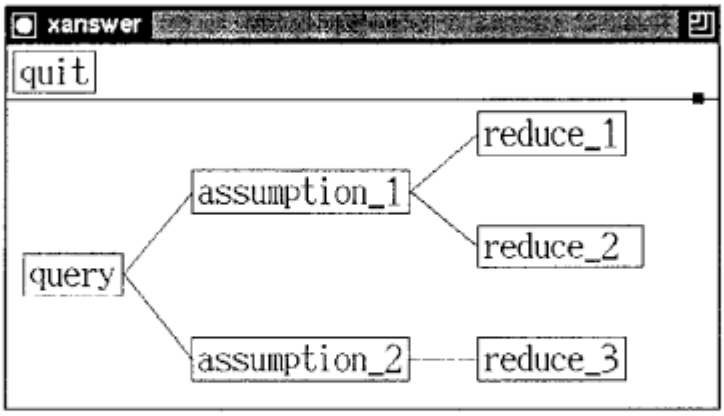

図 5: Answer Window

出木のノードの種類には、query、unit、inherit、merge、lookup がある。query は導出木のルートノードである。unit は通常の導出過程を表す inherit は継承が発生した過程を表す。merge はマージが発生した過程を表す。lookup はルックアップが発生した過程を表す。Answer Window のノードをクリックすると、ノードウィンドウが現れルールが表示される (図 6)。Answer Window を終了する時は「終了」ボタンをクリックする。

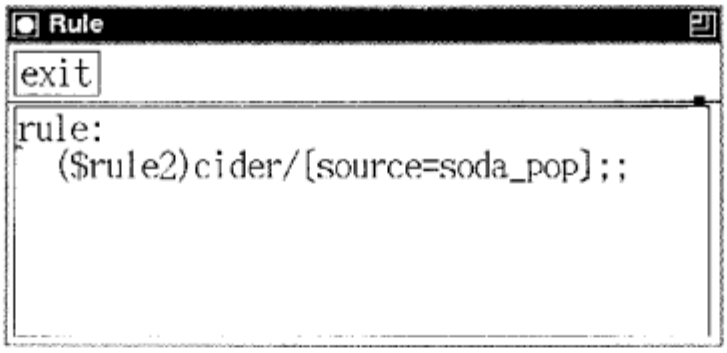

図 6: Node Window

5 *Quixote* サーバ

5.1 PM (Persistence Manager)

5.1.1 概要

本節では、PM 部の概要について述べる。

PM は、*Quixote* プログラムで用いるのライブラリ管理や、*Quixote* プログラムの永続性や、版管理の機能を提供する。*Quixote* 第 3 版では、永続性のための外部記憶としては、非正規型関係データベース Kappa、SIMPOS ファイルシステムおよび、外部の UNIX マシンなどを利用することができる。

5.1.2 機能概要

PM は、以下の機能を持つ。

- *Quixote* プログラムのライブラリ管理
  Quixote では、プログラムの一部をライブラリとして登録し、それを参照することができる。ライブラリプログラムは、PM で管理され、プログラム中で参照が宣言されると、自動的にロードされる。

  PM 内では、複数のプログラムを効率良く管理するために、階層的なディレクトリを定義でき、ディレクトリ毎に分類・格納することができる。また、ディレクトの移動や、ディレクトリ内に格納されたライブラリの検索機能も持っている。

- *Quixote* プログラムの永続性
  Quixote プログラムは、永続性を持っている。即ち、その問合せ処理が終了しても、トップレベルのトランザクションが正常に終了した場合、それまでに行なわれた更新結果が保存されている。PM では、プログラムの表現形式であるテキスト形式、.src 形式、および、.obj 形式の 3 つの形式で格納が可能である。*Quixote* における更新は、問合せ時に、プログラムを付加する方法と、プログラム中の更新規則によるものとがある。更新結果に基づき、最上位のトランザクションの正常終了時に .obj および .src が更新される。

- 版管理
  PM では、更新毎に、それぞれをひとつの版として管理している。そのため、ひとつのデータベース名に対して、版の識別子として、(v.r) を与えている。この版の識別子を指定することにより、任意の版をロードし、それに対して問合せを行なうことができる。

  v は、利用者がプログラムの書き換えなどで変更した場合変更されるもので、自然数で表現される。即ち、同じデータベース名で、create_database を行なった場合に、v は 1 増加する。r は、問合せ時に追加されたプログラムや、更新規則の適用の結果更新が発生された場合変更されるものである。後者の場合、問合せ時には、仮説などが追加されることが多く、一つの版から複数の更新を行なった結果を別々に管理したい場合がある。このため、r は、自然数の列で、すでに同じデータベースに対する更新が行なわれていない場合には、r の最終桁が 1 増加する。すでに別の更新履歴がある場合は、r の桁が増やされる。例えば、データベース db の (1.1) の版を更新した場合、(1.2) 版となり、それを更新すると、(1.3) 版となる。このとき、データベース db の (1.1) 版をもとに、別の更新を行なうとその結果は (1.1.1) 版となり、それを更新すれば、(1. 1.2) 版となる。(図 7 参照。)

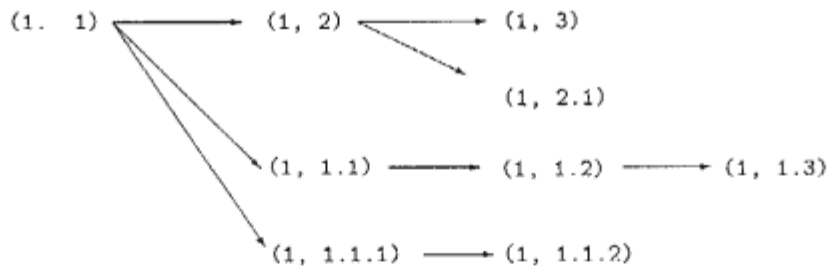

図 7: PM での版管理

- 外部記憶システムとのインタフェース
  PM での永続オブジェクトの格納場所としては、非正規型データベースシステム Kappa や、SIMPOS のファイルシステムを利用できる。さらに、TCP/IP のインタフェースにより接続された、他のマシンへの格納も可能にしている。現在は、TCP/IP インタフェースを用いて他のマシン上の UNIX ファイルシステムを外部記憶として利用するものが実装されている。

5.1.3 モジュール構成図 (概要)

PM では、上記の機能を実現するために、以下の 6 つのモジュールより構成される。

- PM プロセス
- TCP/IP 接続部
- SIMPOS 接続部
- Kappa 用コマンド実行部

- SIMPOS 用コマンド実行部
- UNIX 用コマンド実行部

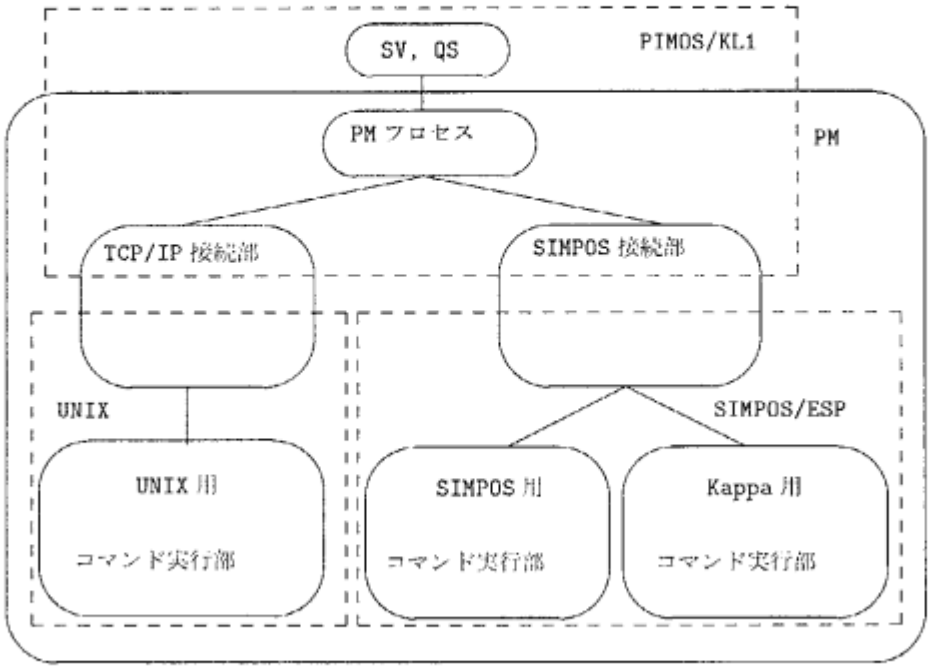

図 8: PM モジュール構成図

PM プロセスおよび、SIMPOS 接続部、TCP/IP 接続部の一部は、PIMOS 上に KL1 言語で実装した。SIMPOS 接続部の一部、Kappa 用コマンド実行部、SIMPOS 用コマンド実行部は、SIMPOS 上に ESP 言語で実装した。UNIX 用コマンド実行部は、UNIX 上に C 言語で実装した。

5.1.4 各モジュールの機能概要

- PM プロセス
  PM プロセスは、SV、QS とストリームで接続され、PM への要求を受け付ける。上位からの要求に従い Kappa、SIMPOS および UNIX ファイルシステムの切り替えを行なう。PM への要求を、切り替えられた外部記憶の指定に従い、SIMPOS 接続部または、TCP/IP 接続部に送る。
- SIMPOS 接続部
  データの格納に、非正規データベース Kappa や SIMPOS を用いる場合、データを SIMPOS 側との通信を行なう必要がある。このために、SIMPOS 接続部は、KL1 部と ESP 部の 2 つから構成される。また、KL1 の内部形式を通信および格納可能な形式へ変換も行なう。
- TCP/IP 接続部
  データの格納に、UNIX 上のファイルシステムを用いる場合、データを UNIX 側との通信を行なう必要がある。このために、TCP/IP 接続部は、KL1 部と C 部の 2 つから構成される。また、KL1 の内部形式を通信および格納可能な形式へ変換も行なう。
- Kappa 用コマンド実行部
  本モジュールでは、非正規データベースシステム Kappa を利用して、プログラムやデータを格納／管理する。PM への要求は、Kappa の機能を用いて本実行部により実行される。このため、階層ディレクトリを表現する関係、検索のための情報を管理する関係、プログラムなどを格納する関係などを Kappa のなかで管理している。
- SIMPOS 用コマンド実行部
  本モジュールでは、SIMPOS ファイルシステムを利用して、プログラムやデータを格納／管理できるようにする。PM への要求は、SIMPOS ファイルシステムを利用して本実行部により実行される。

- UNIX 用コマンド実行部
  本モジュールでは、TCP/IP で接続された UNIX マシン を外部記憶として利用して、プログラムやデータを格納/管理できるようにする。PM への要求は、UNIX マシン 上で本実行部により実行される。

5.2 QD (*Quixote* Data manager)

5.2.1 概要

本節では、QD 部の概要について述べる。QD 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。

QD 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 3 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 QD 内部で行なわれている。

5.2.2 モジュール構成図 (概要)

QD 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 7 つのサブモジュールからなる構成になっている。

- ディスパッチャサブモジュール (DP)
- プログラム管理サブモジュール (PC)
- 束管理サブモジュール (LM)
- プログラムテーブル管理サブモジュール (PTM)
- 更新管理サブモジュール (UM)
- ID 管理サブモジュール (IDM)
- spy 情報管理サブモジュール (SPM)

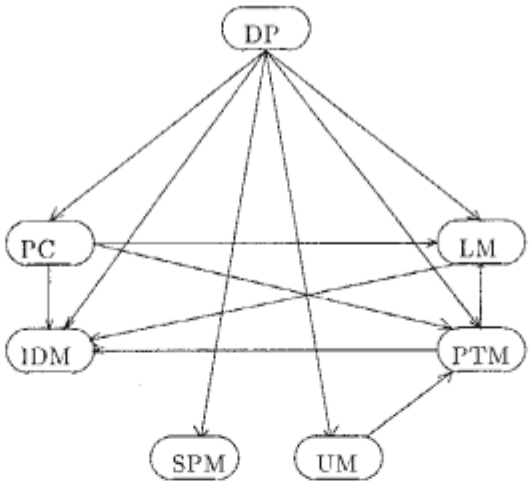

図 9: QD モジュール構成図

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。

5.2.3 各モジュールの機能概要

1) DP
   QD 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QD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、

- QD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、
- QD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、
- ディスパッチの際、必要であれば、PCを用いて、データの変換、逆変換を行なう、
- QD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、トランザクション情報、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情報、および、外部メッセージの既定値などを保持している。

2) PC

PCは、QDに送られるプログラムを内部のデータ形式に変換し、トランザクション単位に保持するサブモジュールである。PCの機能は以下である。

- SVからのプログラム／問合せを入力として、QDの内部形式へ変換する。
- 変換されたプログラムや問合せから関連した情報を、LM、PTMへ送る。
- 問合せの解（および各種エラー）を逆変換を行なって、SVに返す。
- プログラムをトランザクション単位で管理する。
- UMからの情報をもとに、プログラムの更新を行なう。
- SVからの要求に応じて、プログラムを返す。

上記の機能を実現するために、トランザクション識別子から、そのときのプログラムを引けるテーブルを保持している。

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、QC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。

- 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、束を構成するために、必要であれば、中間ノードとして、基本オブジェクトを新たに付加しする。新たに作成した中間ノードはPCに返す。
- QCからの要求に従い、束演算(meet,join,subsume)を行なう。
- SVからの要求(show_lattice)に従い、束構造を返す。
- SVからの要求(query)に従い、束構造を変更する。

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。

4) PTM

PTMは、*Quixote*プログラムの情報を主にインタプリタでの推論処理に適したデータ形式に変換し、QE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。また、ファクトを含むルールの検索や、モジュール間順序に関する問合せなどの提供機能は、PTMで処理される。

このために、以下のテーブルを保持している。

- ルールを管理するルールテーブル
- モジュールを指定して、ルールを検索できる、モジュール索引テーブル
- モジュール間の順序を管理するテーブル

PTMは、4つのサブルーチンによって、上記の機能を実現している。それらは、

- ルール変換処理サブルーチン(RT)
- ダグ化サブルーチン(DAG)
- モジュール順序管理サブルーチン
- ルール検索サブルーチン(GR)

QUIXOTE プログラムは、プログラムを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が行なわれる。この継承処理は、推論処理の前に行なうことができる。従って、PTM では、ルールを内部形式に変換するとともに、モジュール継承処理を行なう。このために、ルール変換サブルーチン (RT) が用いられる。即ち、RT は、以下の機能を持つ。

- PC により変換されたプログラムよりルールテーブル、モジュール索引テーブルを生成する。このとき、モジュールの継承関係によるルールのコピー、ルールの内部形式への変換、DAG 化ルーチンの機能による、ルールの下向き継承、上向き継承関係の DAG 作成をおこなっている。
- PC により変換されたルールより CS の機能を用いて制約解消を行ない、正規化したルールを生成する。
- PC により変換されたルールから内部形式のルールへの変換および逆変換を行なう。
- 更新ルールの実行による更新ファクトのルールテーブルへの反映を行なう。

ルール継承情報作成のために、RT は、ルールヘッドと基本オブジェクトの順序から、ダグ構造を作り出すダク化サブルーチン (DAG) を利用する。DAG は、ルールヘッドのテンプレートの大小関係より、上向き継承用、下向き継承用ルール ID のリストを作成する機能をもつ。

また、モジュール間の順序は、モジュール順序管理サブルーチンが管理しており、モジュールサブモジュール関係の問合せは、モジュール順序管理サブルーチンにて処理される。

ルール検索サブルーチン (GR) は、QE からの要求に従い、RT の作り出したテーブルを用いて、必要なルールの検索を行なう。

5) UM

QUIXOTE では、問合せ時にプログラムを追加することによる更新と、プログラムで、更新述語を用いることによる更新の 2 種類がある。UM では、後者のプログラム中の更新述語による更新情報の管理を行なう。

このため、UM は以下の機能を持つ。

- QE からの更新要求を記録する
- ルール検索時の更新確認の要求に答える

6) IDM

IDM は、

- 基底オブジェクト項と基底ドット項の管理、
- 識別子の管理、
- 基底項の内部データに識別子の付加、
- LM が作成した中間ノードの管理、

などの機能を持ち、QC、QE 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、

- ID からオブジェクト項の内部データ構造を引けるテーブル、
- ID からドット項の内部データ構造を引けるテーブル、
- オブジェクト項の構造から、似た構造を持つオブジェクトの識別子が引けるテーブル、
- モジュールとオブジェクト項から、それらをもつドット項の識別子が引けるテーブル

を保持している。

7) SPM

SPM はトレースのための情報の管理を行なうサブモジュールである。SPM は以下の機能を持つ。

- データベース・セッション時のコマンドに応じて、トレースのための情報を登録、変更、削除を行なう
- DP からの要求に応じて、問合せ開始時に QE へ渡すトレースのための情報を返す

SPM で管理する、トレースのための情報には以下のものがある。

- サブゴール情報
  スパイを行なうサブゴールの情報、およびそのサブゴールのスパイの状態の情報である。スパイの状態には enable と disable がある。

- ルール
  スパイを行なうルールの情報、およびそのルールのスパイの状態の情報である。スパイの状態には enable と disable がある。
- モジュール
  ルール全部のスパイを行なうモジュールの情報、およびそのモジュールのスパイの状態の情報である。スパイの状態には enable と disable がある。
- ゲート
  トレース中に設けられている各ゲートでトレースを行なうか行なわないかの情報である。
- トレース・モード
  トレーサの実行状態の情報で、trace、notrace、spy がある。

5.3 QC (*Quixote* Constraint handler)

5.3.1 QC のモジュール構成

QC は、制約解消系 CS (Constraint Solver) と SD (Solution Description) マネジャから構成される。CS は、おもに、*Quixote* 項間の制約解消の機能を提供する。SD マネジャは、解を表現するための SD というデータ構造の管理、操作を行なう。SD または、ドット項テーブル（ドット項の制約の集合）に対する、add、merge、subsume などの個々の処理の中で、CS の提供する機能を使う。

5.3.2 制約解消系 CS

1) *Quixote* の制約
   Quixote の制約には、以下のものがある。

{ 項. =<, 項 }	包摂 (subsume) 制約
{ 項. ==, 項 }	同値 (cong) 制約
{ 変数. 項 }	単一化子: 変数が項にバインド
{ 変数.{small 制約.large 制約 }. 項 }	単一化子: 制約を持つ変数が項にバインド

2) 制約解消
   制約の集合 S を制約解消するとは、S の中の制約を以下の規則で書き換え、

   - それ以上書換えのできない形になれば制約解消成功
     この場合、制約解消途中でできた単一化子を答として返す。
   - 矛盾が検出されたら制約解消失敗

   とすることである。

$T == T(T:項)$	→	$\emptyset$
$V == T(V:変数.T:項)$	→	V の全ての出現を T で置換 (⇒ 単一化子 {V.T})
$p[l1 = T1, ..., ln = Tn] == q[f1 = U1, ..., fn = Un]$	→	$p == q.\ i \in [1..n]$ に対して、$li == fi.Ti == Ui$
上記以外の $X == Y(X.Y:項)$	→	矛盾
$T =< T(T:項)$	→	$\emptyset$
$X =< Y.Y =< X(X.Y:項)$	→	$X == Y$
$X =< Y.Y =< Z(X.Y.Z:項)$	→	$X =< Y.Y =< Z.X =< Z$
$X =< A.X =< B(X:項.A.B:基底オブジェクト項)$	→	$X =< meet(A.B)$
$A =< X.B =< X(X:項.A.B:基底オブジェクト項)$	→	$join(A.B) =< X$
$B =< A(A.B:基底オブジェクト項)$	→	bsubsume(A.B) が yes なら $\emptyset$、no なら矛盾
$A[...] =< B(A.B:基底オブジェクト項)$	→	$A =< B$
$A =< B[...](A.:基底オブジェクト項)$	→	矛盾
$A[l1 = T1, ..., ln = Tn] =< B[f1 = U1, ..., fm = Um]$	→	$A =< B.$ $\forall j \in [1..m].\ \exists i \in [1..n].\ li = fj.Ti =< Uj$

3) 制約解消系の実装

(a) 変数制約の格納法

制約解消の結果は、変数に関する制約として変数環境ベクタに格納される。変数環境ベクタの値は以下の2種類である。

- 項
- {small 制約,large 制約 }

項は $X == a$ のような等号制約が、単一化にて単一化子 {X,a} となった結果 {small 制約,large 制約 } において、small 制約、large 制約は項のリストであり、それぞれその変数よりも、小さい項、大きな項を表す。例えば、ある変数 X に $a =< X, X =< b$ のような制約がかかった結果、その変数環境ベクタの値は {[a],[b]} となる。

(b) 制約解消系の実装

- 制約の分類

個々の制約は、制約プールの中に格納される。制約プールは現在、次の4つのレベルに制約を分類し、レベルの低いもの (簡単な制約) から処理を行なっている。

- レベル 0

{_,_}	単一化子	→なにもしない
{X,_,X}	単一化子	→なにもしない
{&bot, =<, _}	包摂制約	→なにもしない
{_, =<, &top}	包摂制約	→なにもしない
{_,{_,_},Atom}	単一化子	
{_, ==, _}	同値制約	
{Var, =< 1, Gnd}	包摂制約 (Var の small 制約は見ない)	
{Gnd, 1 =<, Var}	包摂制約 (Var の large 制約は見ない)	
{Type1, =<, Type2}		

- レベル 1

{_,{_,_},Type}	単一化子
{_, =<, _}	包摂制約

- レベル 2

{_,{_,_},Var}	単一化子
{Var1, =< ,Var2}	包摂制約

- レベル 3

{&freeze,_,_,_}	特殊制約 (後回しにしたもの)

- 制約解消系の構成

制約書き換え系では、制約プールから一つ制約をもらうと、その形により以下の操作を行なう。制約がプールから全てなくなったら制約解消成功。 unify 失敗、包摂失敗時には制約解消失敗。

- 各制約の解消操作

制約解消操作には、制約の形に応じて、以下の操作がある。

同値制約
- 単一化制約
- 包摂制約

5.3.3 SD マネジャ

1) 概要

SD は、解を表現するデータ構造であり、インタプリタが実行中に、途中結果を格納するために、用いられる。解は、結論とそれを導く仮定からなるが、仮定も結論も、ドット項に対する制約と変数に対する制約の集合である。SD マネジャは、ドット項制約と変数環境の管理を行ない、SD に対する操作機能 (SD 同士の merge、包摂関係の比較、SD へのドット項制約の付加、解の取り出しなど) をサブルーチンの形で提供する。これらの個々の処理の中で、CS の提供する機能が使われる。第2版で実装した *QUIXOTE* 処理系を評価・検討して、第3版は、SD に関しておもに、以下のような変更を行なった。

- データ構造の変更

第2版は、PIMOS の提供する multiply_keyed_bag を用いて、ドット項に対する制約の集合を実装していた

が、第3版は、この方式をやめて、KL1 の基本的なデータ構造（リストとベクタ）を用いて、SD を表現した。

- SD マネジャの実現方式
  第2版は、SD をプロセスとしていたが、第3版は SD に対する操作機能をサブルーチンとして提供した。
- add 処理の条件判定の変更
  条件判定に、ドット項間の関係を考慮に入れた。

2) データ構造

(a) SD の構成

SD は、仮定部、結論部、累積制約部、変数環境の4項組
{< 仮定部 >,< 結論部 >,< 累積制約部 >,< 変数環境 >}
である。ここで、仮定部は、解の仮定となっている、ドット項に関する制約の集合、結論部は、解の結論となっている、ドット項に関する制約の集合、累積制約部は、ルールのボディ部に含まれる制約を累積するためのものである。また、変数環境は、変数にかかる制約を表し、その変数がドット項変数であれば、対応するドット項表現を格納している。仮定部、結論部、累積制約部は、ドット項テーブル（下記の「ドット項テーブルの構造」参照）として実装されており、変数環境は、変数環境ベクタとドット項参照ベクタの対である（2 節の構文・データ構造」参照）。

(b) ドット項テーブルの構造

ドット項テーブルは、処理効率の観点から、ドット項の形によって、基底ドット項の制約の集まり、パラメトリックドット項の制約の集まり、変数ドット項の制約の集まりに分類される。ここで、基底ドット項とは、ドット項の中に変数を含まないものであり、一意的に定められた識別子を割り振られている。変数ドット項は、ドット項を構成するモジュールまたはヘッドが変数であるものである。それ以外の、変数を含むドット項をパラメトリックドット項という。変数に具体的な値が束縛されることにより、変数ドット項がパラメトリックドット項や基底ドット項に、パラメトリック ドット項が基底ドット項になることがある。このようなとき、変数ドット項テーブルからパラメトリック ドット項テーブルや基底ドット項テーブルに、パラメトリック ドット項テーブルから基底ドット項テーブルに、ドット項のエントリを移す操作が必要である。この操作をドット項テーブルの shallow という。

ドット項テーブルのデータ構造を以下に詳細に述べる。ドット項テーブルは、基底ドット項テーブル（GDT）、パラメトリック ドット項テーブル（PDT）、変数ドット項テーブル（VDT）の3項組である（{GDT.PDT.VDT}）。

- 基底ドット項テーブル（GDT）
  基底ドット項テーブルは、基底ドット項とそれに対応する値の組からなるリストであり、各々の基底ドット項に一意的に付けられるドット項識別子によって、ソートされている。
- パラメトリック ドット項テーブル（PDT）
  パラメトリック ドット項テーブルは、キーとパラメトリック ドット項リストの組のリストである。キーは、処理の対象となるドット項の探索効率向上のために付けられたもので、モジュール テンプレート、ヘッド テンプレート、ラベルの3つ組として実装されている。パラメトリック ドット項リスト（PDL）は、変数リスト、パラメトリック ドット項とそのパラメトリック ドット項に対応する値の3つ組 のリストである。ただし、変数リストは、このパラメトリック ドット項に含まれる変数のリストである。
- 変数ドット項テーブル（VDT）
  変数ドット項テーブル（VDT）は、変数ドット項とそれに対応する値の組のリストである。

3) 機能

SD マネジャは、SD へのドット項制約の付加、SD 同士の merge と比較、解の取り出しなどの機能をサブルーチンとして提供する。SD に対する操作を行なう時、その直前で、SD の shallow を行なっている。これにより、SD が矛盾 inconsistency を含んでいれば、この時点で検出される。SD マネジャが提供する、SD に対する諸操作のうち、主なものについて機能の概略を述べる。

(a) add 処理

add 処理としては、ドット項制約を仮定部に付け加えるものと、結論部に付け加えるものがある。

- SD の仮定部へのドット項制約の付加
  与えられたドット項制約が、結論部に含意されない場合、仮定部にそのドット項制約を付け加える。この機能は、述語

```
add_assumption_if_not_satisfied(QD,^NewQD,ProcMode,SD,DotTable,^Delay,Last,
                                ^NewSD,^Result,^Status)/10.
```

によって実現されている。

- SD の結論部へのドット項制約の付加
  与えられたドット項制約が、結論部に含意されない場合、結論部の対応するドット項の値と単一化する。この機能は、述語

```
add_conclusion_if_not_satisfied(QD,^NewQD,SD,DotTable,^NewSD,
                                ^Result,^Status)/8.
```

によって実現されている。

(b) merge 処理
SD 同士 (SD1 と SD2) の merge を次のように行なう。ふたつの SD の変数環境を継ぎ足して、一つの変数環境とする。このとき、共通の変数は、単一化して同一のものとする。各々の SD の仮定部、結論部、累積制約部同士をドット項テーブルとして merge する。ここで、ドット項テーブルの merge とは、同じドット項表現があれば、その値同士を単一化することである。 wmerge 機能は、述語

```
qxtQE_SD:merge_sd(QD,^NewQD,SD1,SD2,Last,^MergedSD,^RenamingTable,
                  ^Result,^Status)/9.
```

によって実現されている。

(c) subsume 処理
SD 同士の 包摂関係の比較を行なう。 subsume 処理には、仮定部を比較するものと、結論部を比較するものがある。

- SD の仮定部の比較
  SD の仮定部の包摂関係の比較は、述語

```
qxtQE_SD:subsume_assumption(QD,^NewQD,SD1,SD2,Last,^NewSD1,^NewSD2,
                            ^Result,^Status)/9.
```

によって実現されている。

- SD の結論部の比較
  SD の結論部の包摂関係の比較は、述語

```
qxtQE_SD:subsume_conclusion(QD,^NewQD,SD1,SD2,Last,^NewSD1,^NewSD2,
                            ^Result,^Status)/9.
```

によって実現されている。

5.4 QE(*Quixote* Execution)

QE モジュールは、*Quixote* 第二版のインタプリタ・モジュール [成果報告 92] と同等の機能を果たし、さらに機能強化されたものであるが、コードは全面的に書き換えられている。本節では、QE モジュールのサブモジュール構成を明らかにし、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。また、実現した機能を明らかにし、それを実装したアルゴリズムと、*Quixote* 第二版との差異について簡単に述べる。

5.4.1 概要

QE の機能
QE は、QD からサブルーチン呼びだしされ、ルールを駆動し、QC の述語を利用しながら、query に対する解を導出する。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**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。ルックアップを用いることにより、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る。

- 同一のオブジェクト項に関する解は、制約解消を行ない、一つに**併合**する。
- query で**継承**処理を行なうように指示された場合、包摂関係にあるオブジェクト項に関するルールを同時に起動し、継承規則にしたがって制約解消を行なう。
- *Quixote* のルールは、逐次的に記述しない限り、サブゴールの前後関係は解に無関係である。そこで、先行するサブゴールで束縛されていなかった変数が、後続のサブゴールで束縛されるケースでも正しく評価が行なわれるように、そのような変数については**遅延束縛処理**を行なう。

QE のモジュール構成

QE は、主に以下の 4 モジュールから構成される。

- **通常展開部**

  QE の主処理モジュールであり、通常の展開処理 (導出木の生成、ルックアップ、継承、遅延束縛) を行なう。

- **解併合部**

  同一のオブジェクト項に関するルールから得られた解どうしを併合する処理を行なうモジュールである。

- **逐次展開部**

  Quixote のルールのうち、特に逐次展開を行なうように指定されているルールについては、更新処理、トランザクション処理、一貫性制約評価処理などの特殊な処理を記述することができる。逐次展開部は、逐次展開を行なうように指定されたルールの特殊な展開処理を行なうモジュールである。

- **トレーサ**

  QE の実行を対話的にトレースする機能を提供するモジュールである。

第 3 版の機能

第 3 版のための開発作業は、以下の通り。

- *Quixote* 第二版のインタプリタ・モジュールでは、AND 並列アルゴリズムを採用していたので、導出木の形状が展開時の変数の束縛に依らず一定であった。この性質を利用し、導出木を解併合プロセスの集合であり、展開部がこれを起動してまわるアルゴリズムを採用していた。しかし、*Quixote* 第二版では、効率の良い導出を行なうために、AND 逐次アルゴリズム (先行するサブゴールで生じた変数への束縛が後続のサブゴールの展開時に影響を与える) を採用したので、導出木の形状は展開時の変数の束縛に依って動的に変化するようになった。このため、導出木を表現するデータ構造を保持して展開を行なうプロセスが唯一つ存在するアルゴリズムに変更した。この変更を行なうために、QE のコードは全面的に書き換えられた。
- *Quixote* 第二版では導出木の任意の位置に対するルックアップが許されていたが、コーディングの複雑さに比べて効率上必ずしも向上しないので、祖先サブゴールに対するルックアップに限るように変更された。
- *Quixote* 第二版では、データ構造が全面的に変更されたので、QE 全体に関して、これを反映した。
- *Quixote* 第二版のインタプリタ・モジュールの処理で最も時間がかかった部分は、解間の関係を求める部分であった。*Quixote* 第三版では、高速化をはかるため、解に ID を付与し、一度求めた解間の関係はテーブルに登録するアルゴリズムを採用し、解間の関係を求める計算の再計算を防ぐ工夫を行なった。
- *Quixote* 第二版では逐次処理ルールの処理の実装は最低限度のものであったが、*Quixote* 第三版では、機能強化が行なわれ、更新処理、トランザクション処理、一貫性制約の評価処理が実装された。
- QE の実行の様子をトレースするためのトレーサが実装された。

5.4.2 通常展開部

導出木の構造

導出木は、次のような形状の木である。

- 節には、ルール節とサブゴール節の 2 種類がある。
- 根があり、それはサブゴール節である。

- サブゴール節の子節は (もしあるなら)、ルール節である。単一のサブゴール節が複数のルール節の親節となって良い。複数のサブゴール節が共通のルール節の親節になることはない。

  この関係は、あるサブゴールを展開するのに、複数の単一化可能なルールが用いられる関係に対応している。

- ルール節の子節は (もしあるなら)、サブゴール節である。単一のルール節が複数のサブゴール節の親節となって良い。複数のルール節が共通のサブゴール節の親節となることはない。同一のルール節の子節であるサブゴール節どうしの間には、ある関係があり、その関係にしたがって、それらは木を作っている。

  この関係は、あるルールを展開するのに、その複数のサブゴールを展開する関係に対応している。子節であるサブゴール節で作られる木の、根からの距離が等しいサブゴール節は、同一のサブゴールを異なる環境で展開している。

- ルール節、サブゴール節とも、葉となり得る。

サブゴール節の展開アルゴリズム

述語 `qxtQE_sn: expand_a_subgoal/11` は、あるサブゴール S を、ある環境 E のもとで展開する。そのアルゴリズムは、以下の通りである。

1) **ルックアップ条件検査**
   ルックアップ条件を満たしているかどうか検査する。ルックアップできるなら、ルックアップ処理を行ない、以降の手順は踏まない。

2) **get_rule_with_SD**
   引数で与えられた、このサブゴールで展開すべきルールから、"単一化可能かもしれないルール" をすべて取り出し、ヘッドの単一化と、制約の簡約化を行なう。これに成功したルールだけ、以下の手順を行なう。

3) **単一化したルールを一本展開**
   単一化を行なったルールを一本取り出し、展開する。取り出すルールがなくなったら**解の併合**に飛ぶ。

4) **継承ルールを展開**
   そのルールの結果に継承によって影響を与えるルールを展開する。

5) 3. に戻る。

6) **解の併合**
   各ルールから得られた解を、解併合部 (5.4.3節) の機能を用いて併合する。

ルール節の展開アルゴリズム

述語 `qxtQE_rn: expand_a_rule/10` は、あるルール R を、ある環境 Es のもとで展開する。そのアルゴリズムは以下の通りである。

1) **ルール・クラスの検査**
   非逐次展開ルールから逐次展開ルールが呼び出されていないかどうか検査する。失敗したら、実行時エラーとなる。

2) **ボディの展開**
   ルールのボディを先頭から順に展開して行く。

3) **ボディのドット項制約**
   得られた各解が、ルールのボディのドット項制約を満足しているかどうか検査する。満足されていない制約は、その解に対する仮定となる。

4) **ヘッドのドット項制約**
   ルールのヘッドに現れるドット項制約を各解に足し込む。

ルックアップ・アルゴリズム

再帰的なルール展開を正しく停止させるため、既に祖先節で展開したサブゴールを再び子孫節で展開しようとするとき、祖先節の部分結果を子孫節にコピーすることを繰り返して解を求める。これをルックアップという。

あるサブゴール節でサブゴール s を展開する際、その祖先節であるサブゴール節でサブゴール s' を展開していて、$s'\theta = s$ となる代入 θ が存在する場合、s から s' へのルックアップが可能である。

ルックアップできる場合には、必ずルックアップを行なう。手順は以下の通り。

1) **事前処理**
ルックアップしていない部分木について展開を行ない、ルックアップされている節でそれらから求められる解を求める。

2) **ルックアップ・コピー**
ルックアップされている節の解をルックアップしている節にコピーする。以前に同じ解をルックアップ・コピーしている場合には、実際にはコピーは行なわず、以前の結果を流用する。

3) **ルックアップ・サイクル**
ルックアップしている節の解から求められる解を求め、それまでに求められている解と合わせてルックアップされている節まで解を求める。

4) **繰り返し**
ルックアップ・コピー 以降を繰り返す。

5) **停止条件**
ルックアップされている節において、前回のルックアップ・サイクルから比べて新しい解が出ていなければ、停止し、そこまでに求められた解をその節の解とする。

継承処理

サブゴールの展開において、そのサブゴールのドット項に影響を与える全てのルールを展開し、ドット項の継承関係から生じる包摂関係を解の SD に反映させることで継承を実装している。

あるサブゴール節の子節のルールの展開の結果得られた解 (複数) について、以下の手順を行なう。

1) 解から一つ取り出し、環境とする。解がなくなったら終り。
2) "継承で影響しそうなルール" から一本取り出す。ルールがなくなったら 1. から繰り返す。
3) 取り出したルールのヘッドと元のサブゴールとが包摂関係にあるという制約を環境に付加する。これに失敗したルールは展開されない。
4) 取り出したルールを環境のもとで展開し、その解を求める。
5) 再帰的に手順の最初から繰り返す。

遅延束縛処理

一つのルールのボディに複数のサブゴールがあって、互いに変数を共有している場合、先行するサブゴールでは束縛が生じずに、後続のサブゴールで束縛が生じることがある。このような場合、先行するサブゴールを展開している途中では、解の併合やボディのドット項制約の評価を正確に行なうことができないので、それらの処理を、当該変数が束縛されるまで遅延させる。これを、遅延束縛処理と言う。

- **遅延状態の発生**
ボディのドット項の制約を評価する際、および、解の併合を行なう際に、そのルールのルール節よりも祖先の節で導入された変数を参照している場合には、それらの処理を中断し、遅延状態に入る。
- **遅延状態での展開**
遅延状態では、ボディのドット項制約の評価は行なわず、どのようなボディのドット項制約を、どのような解に対して評価すべきであったかを記録する。また、解の併合も行なわない。
- **遅延状態の解消**
遅延の原因となった変数が導入されたルールまで遅延状態での展開が終了し、それらの変数の束縛のバリエーションが確定したところで、遅延発生時まで遡って、そのときの各解に、それらの変数の発生した束縛のバリエーションをすべて適用する。その解を用いて、遅延された各ボディのドット項制約の評価と解の併合を順に解決し、もとのルールの解を求める。途中、更に遅延状態が生ずる場合もある。

解 ID の利用による高速化

Quixote 第二版のインタプリタ・モジュールの処理で最も時間がかかった部分は、解間の充足関係を求める部分であった。*Quixote* 第二版では、高速化をはかるため、解に ID を付与し、一度求めた解間の充足関係はテーブルに登録するアルゴリズムを採用した。

5.4.3 解併合部

解併合部は、同一のオブジェクト項に関するルールから得られた解どうしを併合し、解をまとめる機能をもつ。

解の併合

ゴール節において、複数のルールを展開した場合でも、同一のオブジェクトをヘッドにもつ複数のルールが成功した場合、その解はマージして単一の解にする。この処理を解の併合処理と呼ぶ。

Quixote の解 S は、ゴール節に含まれる変数の束縛集合 C と、その束縛を得る際に用いられた仮定 (assumption) の集合 A の組 $S = (A.C)$ で表す。解を併合するとは、二つの解 $s_1 = (a_1.c_1), s_2 = (a_2.c_2)$ があったとき、解 $s_3 = (a_3.c_3)$ を作ることである。ただし、仮定がそれぞれ異なる場合があるので、その場合、仮定の包摂関係によって解のカバーする範囲ごとに解を生成する。

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' が対応していて、S_2' の元がすべて成り立っているとき s の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。また、制約の集合 S が矛盾することを $null(S)$ と書く。

以下に解併合のアルゴリズムを記す。

- $a_1 \sqsupseteq_S a_2$
  a_1 だけが真である場合というのはないので、s_2 は残し、$s_1.s_2$ をマージした解を作って、s_1 は捨てる。
- $a_2 \sqsupseteq_S a_1$
  a_2 だけが真である場合というのはないので、s_1 は残し、$s_2.s_2$ をマージした解を作って、s_2 は捨てる。
- $a_1 \sqsupseteq_S a_2 \wedge a_2 \sqsupseteq_S a_1$
  どちらか一方が真である場合というのはないので、$s_1.s_2$ をマージした解を作成し、$s_1.s_2$ ともに捨てる。
- $\mathrm{null}(a_1 \cup a_2)$ (a_1, a_2 に包摂関係がない)
  両方の assumption が共に成り立つ場合というのはないので、$s_1.s_2$ をそのまま解として残す。
- **上記のどれでもないとき**
  包摂関係はないが矛盾もしない、すなわち、共通部分をもつ場合には、$s_1.s_2$ をそのまま解として残し、さらに、$s_1.s_2$ をマージした合成解を生成する。

充足関係テーブルへの対応

すでに節 5.4.2 で述べた解の間の充足関係テーブルに対応して解併合部でも計算結果の再利用の機能を追加した。

解併合部で対応した機能としては、以下の 2 点である。

- 包摂関係の検査時の充足関係テーブルチェック

  解併合部では、解のまとめ上げをおこなう際に 2 解の間の包摂関係を調べる。これには、SD が提供している包摂関係検査用述語 (subsume_assumption, subsume_conclusion) を使う。しかし、包摂関係の検査は処理が重いので充足関係テーブルを一種のキャッシュの様に使うことで処理の軽減を図った。

  解併合部がおこなう処理としては、2 解の間の関係を調べる際に、まずはじめに充足関係テーブルの内容を調べて 2 解の関係がわかれば SD の包摂関係検査述語を呼ばないようにする。充足関係テーブルを調べても 2 解の関係がわからないときは、包摂関係検査述語を呼び検査結果を充足関係テーブルに書き込み反映する。

- 解マージの結果の充足関係テーブルへの反映

  解併合部では、解のまとめ上げをおこなう際に 2 解をマージして新たな解をつくる。マージによってもとの 2 解と新たにマージしてできた解との間には包摂関係があるので、この情報を充足関係テーブルに書き込み反映する。

5.4.4 逐次展開部

逐次展開部は、インタプリタ (IP) を構成するモジュールの一部で、通常展開部から呼び出される。

逐次展開部は、トランザクションを記述することができる更新用のルール (以降更新ルールと呼ぶ) の展開をおこなう機能をもつ。

更新ルールは、

$$head \Leftarrow cluster; cluster; \ldots ;;$$

という形式のルールで、通常ルールとの違いはボディ部が；で区切られたクラスタという単位の並びになっている点である。

クラスタには、サブゴールの他に、トランザクション・ポイントを指定するためのトランザクション・コントローラや、更新処理をおこなうためのアップデート・コントローラが書ける。逐次展開部は、通常展開部からクラスタ毎に呼び出されて、トランザクション・コントローラとアップデート・コントローラの展開処理をおこなう。

これらの展開機能について以下に機能毎に記述する。

1) トランザクション・コントローラの展開

以下にトランザクション・コントローラ毎の展開機能について記述する。

- &begin_transaction (&bt)
  新たなトランザクション・ポイントを設定する。
  逐次展開部の処理としては、QD から新たなトランザクション識別子を取得し、その時点で求まっている中間解を覚えておいて展開を続ける。
- &end_transaction (&et)
  最も近いトランザクション・ポイントからここまでの間におこなわれた更新を有効化する。
  逐次展開部の処理としては、QD から新たなトランザクション識別子を取得し、その時点で求まっている中間解を新たな解として展開を続ける。
- &abort_transaction (&at)
  最も近いトランザクション・ポイントからここまでの間におこなわれた更新を無効化する。
  逐次展開部の処理としては、QD から新たなトランザクション識別子を取得し、その時点で求まっている中間解を破棄し、&begin_transaction で覚えておいた中間解を新たな解として展開を続ける。
- &consis(i-check)
  i-check に書かれている条件を検査して条件を満たせば &end_transaction 、満たさなければ &abort_transaction として処理する。i-check には、条件としてサブゴール、制約が書ける。
  逐次展開部の処理としては、制約の評価、サブゴールの展開の順に処理をおこない、両方共に成功したら &end_transaction 処理をおこない、少なくともどちらか一方が失敗したら &abort_transaction 処理をおこなう。
- &inconsis(i-check)
  i-check に書かれている条件を検査して条件を満たせば &abort_transaction 、満たさなければ &end_transaction として処理する。i-check には、条件としてサブゴール、制約が書ける。
  逐次展開部の処理としては、制約の評価、サブゴールの展開の順に処理をおこない、両方共に成功したら &abort_transaction 処理をおこない、少なくともどちらか一方が失敗したら &end_transaction 処理をおこなう。

2) アップデート・コントローラの展開

データベースの更新をおこなう。更新するデータベースとしては、QD が保持しているルール／ファクトの集合と IP が展開実行時に保持している中間解 (ドット項制約の集合で SD として保持している) の集合を更新対象とし、両者の間の整合性を保つ。

- +m:o
  データベースにモジュール m の オブジェクト o を追加する。
  QD に m:o を指定してオブジェクト追加を依頼する。
- -m:o
  データベースからモジュール m のオブジェクト o を削除する。データベースに指定したオブジェクトがない場合はデータベースはそのままである。
  QD に m:o を指定してオブジェクト削除を依頼する。その際、削除したオブジェクトに関するドット項の情報を取得し、SD に依頼してそのドット項を中間解から削除する。
- +m:o/[l—a]
  データベース中のオブジェクト m:o についてドット項制約 m:o.l =< a を追加する。データベースにオブジェクト m:o がないときは、オブジェクトを追加しさらにドット項制約を追加する。
  SD に依頼して中間解にドット項制約 m:o.l =< a を追加する。ドット項制約を追加したことによって矛盾が生じたときは、実行時エラーとして処理を中止する。

- -m:oll
  データベース中のオブジェクト m:o の l に関する制約を削除する。データベースに指定したドット項がない場合はデータベースはそのままである。
  QD に m:oll を指定してドット項制約の削除を依頼する。SD に m:oll を依頼してドット項制約を中間解から削除する。

5.4.5 トレーサ

トレーサの機能

トレース機能は、query に対する解を導出する過程中、ルール、サブゴール、クラスタに対する処理を行なう際に、一旦導出を止めて、プログラムのどこに処理が行なわれているかを表示する機能である。トレーサはトレース機能を実現するモジュールである。

ゲートは、処理中に設定されているトレースを行なう地点である。トレーサは指定されたゲートだけをトレースすることができる。ゲートは以下の 9 種類である。

- call ゲート
  call ゲートは、ルール節またはサブゴール節が子節の展開を始める前の段階に設定されているゲートである。
- exit ゲート
  exit ゲートは、ルール節またはサブゴール節が解のまとめ上げを終えた時の段階に設定されているゲートである。
- fail ゲート
  fail ゲートはサブゴール節の展開が失敗した時の段階に設定されているゲートである。
- lookup ゲート
  lookup ゲートは、あるルックアップされる節において、ルックアップの繰り返し検査を行なった直後の段階で、ルックアップが繰り返しと判定された段階に設定したゲートである。
- bt ゲート
  bt ゲートはトランザクション・コントローラ begin transaction を処理する直前に設定されたゲートである。
- et ゲート
  et ゲートは、トランザクション・コントローラ end transaction を処理する直前に設定されたゲートである。
- at ゲート
  at ゲートは、トランザクション・コントローラ abort transaction を処理する直前に設定されたゲートである。
- update ゲート
  update ゲートは、更新処理をする直前に設定されたゲートである。
- ic ゲート
  ic ゲートは、& inconsis または & consis を処理する直前に設定されたゲートである。

トレース・モードは、どのようにトレースを行なうかの状態である。トレーサは指定されたトレース・モードでトレースすることができる。トレース・モードは以下の 3 種類である。

- notrace
  一切トレースを行なわない。
- trace
  指定されたゲートを通過する時にトレースを行なう。
- spy
  スパイ指定したルール、サブゴールのみ、指定されたゲートを通過する時にトレースを行なう。

またトレーサはトレース中に以下のコマンドを実行させる事ができる。

- インスペクト
  インスペクトとは解を解析することである。インスペクトのコマンド inspect はトレースされている時の解の中よりユーザより指定された箇所を表示させる。
- 今後のトレースの指定
  今後のトレースの今後のトレースの指定のコマンドには step、spy、notrace, skip の 4 種類がある。それぞれのコマンドの働きは以下である。
  - step
    コマンド step は次のトレースを、指定されているゲートを次に通過する時に行なわせる。

spy
コマンド spy は次のトレースを、スパイ指定されているルールまたはサブゴールが指定されているゲートを通過する時に行なわせる。

notrace
コマンド notrace はそれ以降トレースを行なわなくさせる。

skip
コマンド skip は次のトレースを、現在トレース中のルールまたはサブゴールが exit ゲートを通過する時に行なわせる。このコマンドは call ゲートをトレースしている時のみに実行させることができる。

- 実行系の中断
  実行系の中断のコマンド abort を行なえば、解の導出を途中でやめる。

トレース処理

トレースに必要な情報には以下のものがある。

- トレース・パラメータ
  解の導出の開始時に QD より渡される情報である。トレース・パラメータは以下の 3 種類の情報からなる。
  - 各ゲートでトレースを行なうか否かの情報
  - トレース・モード
  - スパイ指定されているサブゴールのリスト
- 処理中のルール、サブゴール、またはクラスタ
- その時点で求まっている解

トレース処理の手順は以下の通りである。

1) 解の導出中ゲートを通過する時、その時行なわれている処理はトレーサにトレースに必要な情報を渡し、トレーサの処理を開始させる。トレーサの処理が終了するまで、その時の処理は一時的に止まる。
2) トレーサはゲートの情報よりその時のゲートでトレースするか否かを判断する。トレースしないと判断した場合は、トレーサは処理を終了する。トレースすると判断した場合は、処理を次へ進める。
3) トレーサはトレース・モードを見て処理を決める。トレース・モードが notrace の場合は、トレーサは処理を終了する。トレース・モードが trace の場合は、処理を次へ進める。トレース・モードが spy の場合は、下のそれぞれの場合の処理を行なう。
   - トレースするものがクラスタの場合は、トレーサは処理を終了する。
   - トレースするものがサブゴールの場合は、スパイ指定されているサブゴールの中にトレースするサブゴールと同じものがあるかを調べる。同じものがある場合は、トレーサは処理を終了する。同じものがある場合は、処理を次へ進める。
   - トレースするものがルールの場合は、ルールのスパイ・フラグが on か off かを調べる。off の場合は、トレーサは処理を終了する。on の場合は、処理を次へ進める。
4) トレーサは QD へトレースの発生を伝えるメッセージを送り、トレースするルール、サブゴール、またはクラスタを qshell または qmacs に表示してもらう。
5) トレーサはコマンドを待つ状態になる。

インスペクト

インスペクトのコマンド inspect の処理手順は以下の通りである。

1) コマンド inspect が来る。
2) トレーサはユーザに、解が複数ある場合は解のどれをみるか、また解の仮定、結論、変数制約のどれを見るかを指定させる。
3) トレーサは処理開始時に渡された解より、指定された箇所を取り出す。
4) トレーサは取り出した箇所を QD へ渡し、表示してもらう。

今後のトレースの指定

トレーサは今後のトレースの指定のコマンドを、トレース・モードを変更させることによって実現している。今後のトレースの指定のコマンド別のトレース・モード変更方法は以下の通りである。

- step
  トレース・モードを trace にすることで実現する。
- spy
  トレース・モードを spy にすることで実現する。
- notrace
  トレース・モードを notrace にすることで実現する。
- skip
  トレースしている節のトレース・モードを trace、その節の子孫節のトレース・モードを notrace にすることで実現する。

トレースの継続のコマンドを受けた後、トレーサは処理を終了する。

実行系の中断

実行系を中断させるコマンドを受けとった時、トレーサは、トレースを開始させた処理へ中断のコマンドを受けとった事を知らせる。そしてトレーサは処理を終了する。

QE はその後、処理系を中断する。

6 *Quixote* のコマンド

本節は *Quixote* 第3版の Qmacs (*Quixote* Emacs) の使用手引である。

6.1 準備

ss151 を使用する場合と ME を使用する場合と異なっているので注意が必要である。実際の設定に当たっては ss151:/db/Quixote/v3/src/QF/cmacs/lisp/README の内容をカットアンドペーストされたい。

6.1.1 ss151 を使う場合

- 環境変数 QXTDIR の設定
  Quixote ルートディレクトリを環境変数 QXTDIR に設定する。csh を使用している場合には、$HOME/.cshrc に以下を設定する。

```
# Quixote
setenv QXTDIR /db/Quixote/v3/SS
```

- $HOME/.cmacs の設定
  以下の行を $HOME/.cmacs に入れる。

```
;; quixote for SS
(setq exec-path (cons (concat (expand-file-name
       (substitute-in-file-name "$QXTDIR")) "lib") exec-path))
(setq load-path (cons (concat (expand-file-name
       (substitute-in-file-name "$QXTDIR")) "/lib/emacs")
        load-path))
(autoload 'qxt-start
  "qxt-all" "Start Quixote session to SERVER."
  t nil)
(autoload 'qxt-session-start
  "qxt-all" "Start Quixote session to SERVER."
  t nil)
(setq auto-mode-alist
```

```
      (cons '("\\.qxt$" . qxt-mode) auto-mode-alist))
(autoload 'qxt-mode
  "qxt-mode" "Major mode for Quixote, DOODKRL."
  t nil)
```

6.1.2 PSI-III を使う場合

- 環境変数 QXTDIR の設定
  Quixote ルートディレクトリを環境変数 QXTDIR に設定する。auspex の /db を ME の /db に NFS していて、csh を使用している場合には、$HOME/.cshrc に以下を設定する。

```
# Quixote
setenv QXTDIR /db/Quixote/v3/ME
```

  auspex の /db を PSI-III の xxx に NFS している場合、QXTDIR を次のように設定する。

```
# Quixote
setenv QXTDIR xxx/Quixote/v3
```

- $HOME/.emacs の設定
  以下の行を $HOME/.emacs に入れる。

```
;; quixote for ME
(setq exec-path (cons (concat (expand-file-name
       (substitute-in-file-name "$QXTDIR")) "lib") exec-path))
(setq load-path (cons (concat (expand-file-name
       (substitute-in-file-name "$QXTDIR")) "/lib/emacs")
        load-path))
(autoload 'qxt-start
  "qxt-all" "Start Quixote session to SERVER."
  t nil)
(autoload 'qxt-session-start
  "qxt-all" "Start Quixote session to SERVER."
  t nil)
(autoload 'qxt-shell
  "qxt-all" "Start Quixote shell."
  t nil)
(setq auto-mode-alist
      (cons '("\\.qxt$" . qxt-mode) auto-mode-alist))
(autoload 'qxt-mode
  "qxt-mode" "Major mode for Quixote, DOODKRL."
  t nil)
```

6.1.3 サーバの設定

Quixote サーバを下記の場所から持って来てロードする。

```
icpsi587::>sys>user>quixote>Third>SAVE>qxt_xxx.sav
                                      qxt_xxx.ptf
```

もしくは、

```
icpsi251::>sys>user>qxt>qxt>qxt_xxx.sav
                           qxt_xxx.ptf
```

6.2 起動方法

6.2.1 サーバの起動

現状では、PSI の PSEUDO PIMOS V3.5 上でサーバが動作する。PIMOS シェル上で以下を入力すると *Quixote* ウィンドウができる。

```
qifServer <=window("server")
```

6.2.2 サーバの終了

Quixote ウィンドウ上で shutdown を入力する。

6.2.3 クライアントの起動

EMACS 上で次のコマンドを実行する。

```
M-x qxt-start
```
または
```
M-x qxt-session-start
```

ミニバッファ上で Host: と聞いてくるので、ホスト名を入れる。

```
Host:  icpsi210
```

qxt-start コマンドの場合、.emacs に以下を設定しておくと、ホスト名を聞いてこない。

```
(setq qxt-host-name "icpsi210")
```

qxt-session-start コマンドの場合は、ホスト設定があってもホスト名を必ず聞く。

6.2.4 クライアントの終了

EMACS 上で次のコマンドを実行する。Realy Quit ? (yes or no) と確認があるので、yes と入力する。

```
M-x qxt-quit
Realy Quit ? (yes or no) yes
```

6.2.5 クライアントのリセット

異常が発生し、サーバを再起動した場合、emacs の状態を起動時のサーバに合わせなければならない。以下のコマンドを実行する。

```
M-x qxt-reset
```

6.3 *Quixote* モードの使用方法

ファイル名が *.qxt のファイルを入力すると Qxt モードとなる。Qxt モードでは以下のキーアサインを使用できる。

```
C-c C-s qxt-session-start
C-c C-z qxt-quit
C-c C-c qxt-close-database
C-c C-q qxt-query
C-c C-r qxt-create-database
```

6.4 *Quixote* コマンド

6.4.1 セッション共通コマンド

- qxt-set-window-mode
  X ウィンドウを使用するかしないかを設定する。M-x qxt-set-window-mode と入力すると、WindowMode: と聞いてくるので、use または no use を指定する。入力にはコンプリーションが利用できる。use を指定すると、後述する qxt-query.qxt-show-lattice、qxt-show-module-hierarchy の結果を X ウィンドウ上に表示する。この設定は新たに設定するまで有効となる。Qmacs 起動時は no use を設定している。

  .emacs に以下を設定しておくと、起動時の既定値を use にできる。

```
(setq qxt-window-mode t)
```

```
M-x qxt-set-window-mode
WindowMode: use
```

- qxt-set-qmacs-i-mode
  Qmacs の実行モードを設定する。M-x qxt-set-qmacs-i-mode を入力すると、QmacsMode: と聞いてくるので、I または D を指定する。入力にはコンプリーションが利用できる。I を指定すると、パーサとアンパーサを別プロセスで実行する。D を指定すると、Emacs 内部で実行する。この設定は新たに設定するまで有効となる。Qmacs 起動時は D を設定している。

```
M-x qxt-set-qmacs-i-mode
QmacsMode: D
```

  .emacs に以下を設定しておくと、起動時の既定値を I にできる。

```
(setq qxt-qmacs-i-mode t)
```

- qxt-show-qmacs-mode
  Qmacs の現在のモードを表示する。M-x qxt-show-qmacs-mode と入力すると、client-state、current-session、conv-mode、window-mode、qmacs-i-mode の設定値をカレントバッファに表示する。

```
** Qmacs Mode **
  Client State    : nil
  Current Session : server
  Conv mode       : conv
  Window mode     : no use
  Qmacs I mode    : D
-----------------------
M-x qxt-show-qmacs-mode
```

- qxt-to-message
  サーバのプロセスにメッセージを送る *Quixote* 開発者用のコマンドである。プロンプトにしたがって入力する。

```
M-x qxt-to-message
ProcessName: sv
MessageName: process
InputArgument: (a, b)
OutputArgumentCount: 2
```

6.4.2 サーバセッションコマンド

- qxt-create-database
  バッファリージョンの内容を指定したデータベース名で登録する。このコマンドを実行すると、データベースセッションになる。

  リージョンを指定して、M-x qxt-create-database コマンドを入れる。ミニバッファ上で DatabaseName: と聞いてくるので、データベース名を入れる。

```
□ &program;;
  &rule;;
  m_m::o;;
  m_m::o[l_l=1];;
  &end. ■
----------------------
M-x qxt-create-database
DatabaseName: db
```

- qxt-create-database-file
  ファイル名とデータベース名を指定して登録する。このコマンドを実行すると、データベースセッションになる。

  M-x qxt-create-database-file コマンドを入れる。ミニバッファ上で FileName: と聞いてくるので、ファイル名を入れる。次に DatabaseName: と聞いてくるので、データベース名を入れる。

```
M-x qxt-create-database-file
FileName: db.qxt
DatabaseName: db
```

- qxt-open-database
  作成済みのデータベースをオープンする。このコマンドを実行すると、データベースセッションになる。データベース名にはバージョン、リビジョンを DatabaseName.VerNo.RevNo1.RevNo2 ··· と指定する。バージョン、リビジョンを省略したばあい、最新バージョンを open する。ディレクトリを指定しない場合、カレントディレクトリよりオープンする。ディレクトリを指定した場合、カレントディレクトリが指定したディレクトリになる。オープンモードとして read_only または exclusive を指定できる。ファイルタイプとして src または obj を指定できる。

  M-x qxt-open-database コマンドを入れる。DatabaseName: と聞いてくるのでデータベース名を入れる。DirectoryName: と聞いてくるのでディレクトリ名を入れる。複数回指定できる。何も入れず改行すると、ディレクトリ名の入力を終了する。ディレクトリを指定しない場合、何も入れず改行する。OpenMode: と聞いてくるのでオープンモードを入れる。オープンモードには exclusive または read_only を指定する。データベースを読みとり専用か排他的で利用するかを指定する。オープンモードの入力にはコンプリーションが利用できる。オープンモードとの入力で何も入れず改行すると、exclusive 指定となる。FileType: と聞いてくるのでファイルタイプを入れる。ファイルタイプには src または obj を指定する。保存しているデータベースの形式のどちらを利用するかを指定する。obj を指定すると高速となる。ファイルタイプの入力にはコンプリーションが利用できる。ファイルタイプの入力で何も入れず改行すると、obj 指定となる。

```
M-x qxt-open-database
DatabaseName: db.1.2.3
DirectoryName:
OpenMode:
FileType:
```

- qxt-show-status
  現在サーバが保持している *QUIXOTE* プログラムを表示する。

```
M-x qxt-show-status
```

- qxt-cd
  カレントディレクトリを変更する。デフォルトは、ルートディレクトリである。

```
M-x qxt-cd
DirectoryName: dir
```

- qxt-pwd
  カレントディレクトリを表示する。

```
M-x qxt-pwd
```

- qxt-ls
  カレントディレクトリのデータベース名、サブディレクトリを表示する。

```
M-x qxt-ls
```

- qxt-ls-l
  条件にあったカレントディレクトリのデータベースを表示する。オプションとして l または n を指定できる。l を指定した場合、バージョンとリビジョン付きでデータベース名を表示する。n を指定した場合、バージョンとリビジョン付きで最大バージョンのデータベース名を表示する。
  オプション指定にはコンプリーションが利用できる。オプションの入力で何も入れず改行すると、l 指定となる。

```
M-x qxt-ls-l
Option: l
```

- qxt-mkdir
  カレントディレクトリに、ディレクトリを作成する。

```
M-x qxt-mkdir
DirectoryName: dir
```

- qxt-rmdir
  カレントディレクトリより、ディレクトリを削除する。

```
M-x qxt-rmdir
DirectoryName: dir
```

- qxt-rm
  カレントディレクトリより、指定バージョン、リビジョンのデータベース名のファイルを削除する。

```
M-x qxt-rm
DatabaseName: db.1.1
```

- qxt-rm-all
  カレントディレクトリより、指定バージョンのデータベース名のファイルを削除する。バージョン指定を省略した場合、すべてのバージョンのファイルを削除する。

```
M-x qxt-rm-all
DatabaseName: db
Version:
```

- qxt-purge
  カレントディレクトリより、指定バージョンのデータベース名のファイルのうち、最大バージョン以外のファイルを削除する。

```
M-x qxt-purge
DatabaseName: db
```

- qxt-get-text-DB
  指定したデータベースのqxtファイルを読み込む。データベース名にはバージョン、リビジョンを指定できる。DatabaseName.VerNo.RevNo1.RevNo2.... と指定する。バージョン、リビジョンを省略した場合、最新バージョンを読み込む。

  ディレクトリ名は複数回指定できる。何も入れず改行すると、ディレクトリ名の入力を終了する。ディレクトリ名を指定しない場合は何も入れず改行する。

```
M-x qxt-get-text-DB
DirectoryName:
DatabaseName: db
```

- qxt-get-DBswitch
  現在の DB スイッチを表示する。

```
M-x qxt-get-DBswitch
```

- qxt-set-DBswitch
  現在の DB スイッチを設定する。Switch: と聞いてくるので、DB スイッチを指定する。DB スイッチとして simpos、kappa、unix を指定できる。スイッチ入力時にコンプリーションを利用できる。何も指定せず改行すると、simpos が指定される。unix を指定すると、さらに HostName: と聞いてくるので、ホスト名を指定する。

```
M-x qxt-set-DBswitch
Switch: simpos
```

6.4.3 データベースセッションコマンド (1)

- qxt-query
  バッファリージョンの内容でデータベースに問合せる。リージョンを指定して、M-x qxt-query を入力する。

```
   □?- m_m:o[l_l=1]. ■
----------------------
M-x qxt-query
```

- qxt-query-file
  ファイルの内容でデータベースに問合せる。M-x qxt-query を入力する。FileName: と聞いてくるので、ファイル名を指定する。

```
M-x qxt-query
Filename : db.query
```

- qxt-close-database
  データベースをクローズする。このコマンドを実行すると、サーバセッションモードになる。M-x qxt-close-database と入力する。

```
M-x qxt-close-database
```

- qxt-begin-transaction
  トランザクションを開始する。M-x qxt-begin-transaction と入力する。

```
M-x qxt-begin-transaction
```

- qxt-end-transaction
  トランザクションを正常に終了する。

```
M-x qxt-end-transaction
```

- qxt-abort-transaction
  トランザクションをアボートする。

```
M-x qxt-abort-transaction
```

- qxt-show-module
  モジュール関係をカレントバッファに表示する。

```
** module **
[{a, b}, {b,c}]
-----------------------
M-x qxt-show-module
```

- qxt-show-module-nodes
  モジュールノードを表示する。

```
M-x qxt-show-module-nodes
ModuleId: m
Neighbour: 2
```

- qxt-show-lattice
  束を表示する。ウィンドウモードが no use の場合、カレントバッファにに表示する。ウィンドウモードが use の場合、束ウィンドウに表示する。

```
M-x qxt-show-lattice
```

- qxt-show-lattice-nodes
  束ノードをカレントバッファに表示する。

```
M-x qxt-show-lattice-nodes
BasicNode: a
Neighbour: 2
```

- qxt-compress-set
  リージョンで指定した基本オブジェクトのリストに対し compress_set を実行し、結果をカレントバッファに表示する。オプションには minimal と maxmal のいずれかを指定する。オプション入力時にコンプリーションが利用できる。

```
   □ [a, b, c, d] ■
-----------------------
M-x qxt-compress-set
CompressOption: minimal
```

- qxt-delete-database
  データベースを削除する。このコマンドを実行するとサーバセッションモードになる。

```
M-x qxt-delete-database
```

- qxt-show-rules
  指定モジュールのルールを表示する。

```
M-x qxt-show-rules
ModuleId : m
```

- qxt-get-id-rule
  リージョンで指定したユーザルール ID のリストに対応するルールを表示する。

```
   □ ["rule1", "rule2"] ■
-----------------------
M-x qxt-get-id-rule
```

- qxt-get-default-mode
  問合せ時のデフォルトモードの値を表示する。

```
** default mode **
proc == &multi
answer == &normal
inheritance == &all
merge == &yes
explanation == &on
-----------------------
M-x qxt-get-default-mode
```

- qxt-set-default-mode
  問合せ時のデフォルトモードの値を設定する。ModeId: と聞いてくるので、proc、answer、inheritance、merge、explanation のいずれかを入れる。proc を指定した場合、ProcMode: と聞いてくるので、multi または single を指定する。answer 指定した場合、AnswerMode: と聞いてくるので、normal または mininal を指定する。inheritance 指定した場合、InheritanceMode: と聞いてくるので、all、down、up、no のいずれかを指定する。merge 指定した場合、MergeMode: と聞いてくるので、yes または no を指定する。explanation 指定した場合、ExplanationMode: と聞いてくるので、on または off を指定する。
  入力時にはコンプリーションが利用できる。

```
M-x qxt-set-default-mode
ModeId: proc
ProcMode : multi
```

- qxt-show-basic-objects
  指定した種類の基本オブジェクトを表示する。Kind: と聞いて来るので、mid、label、value のいずれかを指定する。

```
M-x qxt-show-basic-objects
Kind: label
```

- qxt-show-objects
  指定したモジュールのオブジェクトをカレントバッファに表示する。

```
M-x qxt-show-objects
ModuleId: m
```

- qxt-show-dot-label
  指定したモジュールのドットラベルをカレントバッファに表示する。

```
M-x qxt-show-dot-label
ModuleId: m
```

- qxt-show-module-hierarchy
  モジュール関係とモジュール別のルールを表示する。ウィンドウモードが no use の場合、カレントバッファにに表示する。ウィンドウモードが use の場合、Module Hierarchy Window に表示する。

  ```
  M-x qxt-show-module-hierarchy
  ```

- qxt-change-conv-mode
  conv-mode を変更する。*Quixote* 開発用のコマンドである。ConvMode: と聞いてくるので、conv_QD、conv、no_conv のいずれかを指定する。ユーザモードでは conv である。起動時の値は conv である。

  ```
  M-x qxt-change-conv-mode
  ConvMode: conv
  ```

  .emacs に以下を設定しておくと、起動時の既定値を変更できる。ただし、サーバ側の起動時の既定値と一致していなければならない。

  ```
  (setq qxt-conv-mode 'no_conv)
  ```
  または
  ```
  (setq qxt-conv-mode 'conv_QD)
  ```

6.4.4 データベースセッションコマンド (2)

主としてトレースに関係したデータベースセッションコマンドである。

- qxt-set-trace-mode
  トレースモードを設定する。TraceMode: と聞いてくるので、notrace、trace、spy のいずれかを指定する。コンプリーションが利用できる。

  ```
  M-x qxt-set-trace-mode
  TraceMode : trace
  ```

- qxt-get-trace-mode
  現在設定されているトレースモードをカレントバッファに表示する。

  ```
  M-x qxt-get-trace-mode
  ```

- qxt-set-gate
  一つのトレースゲートを設定する。GateId: と聞いてくるので、call、exit、fail、lookup、bt、et、at、ic、update のいずれかを指定する。Switch: と聞いてくるので、on または off を指定する。入力にはコンプリーションが利用できる。

  ```
  M-x qxt-set-gate
  GateId : call
  Switch : on
  ```

- qxt-set-gate-all
  すべてのトレースゲートを設定する。Call: Exit: Fail: Lookup: Bt: Et: At: Ic: Update: と順番に聞いてくるので、on または off を指定する。入力にはコンプリーションが利用できる。改行のみを入力すると off を指定したことになる。

  ```
  M-x qxt-set-gate-all
  Call: on
  Exit: on
  Fail: on
  Lookup: on
  ```

```
Bt: on
Et: on
At: on
Ic: on
Update:
```

- qxt-get-gate
  一つのトレースゲートの設定をカレントバッファに表示する。GateId: と聞いてくるので、call、exit、fail、lookup、bt、et、at、ic、update のいずれかを指定する。入力にはコンプリーションが利用できる。

```
M-x qxt-get-gate
GateID: call
```

- qxt-get-gate-all
  すべてのトレースゲートの設定をカレントバッファに表示する。

```
M-x qxt-get-gate-all
```

- qxt-spy-at-subgoals
  サブゴールにスパイを設定する。ModuleId: と聞いてくるので、モジュール ID を指定する。Oterm: と聞いてくるので、オブジェクト項を指定する。サブゴールは複数指定できる。ModuleId: と Oterm: の両方で改行のみを入力するとサブゴールの指定を終了する。

```
M-x qxt-spy-at-subgoals
ModuleId: m
Oterm: o
ModuleId:
Oterm:
```

- qxt-spy-at-rules
  ルールにスパイを設定する。ModuleId: と聞いてくるので、モジュール ID を指定する。RuleId: と聞いてくるので、ルール ID を指定する。ルールは複数指定できる。ModuleId: と RuleId: の両方で改行のみを入力するとサブゴールの指定を終了する。

```
M-x qxt-spy-at-rules
ModuleId m
RuleID: r
ModuleId
RuleID:
```

- qxt-spy-at-modules
  モジュールにスパイを設定する。ModuleId: と聞いてくるので、モジュール ID を指定する。モジュール ID は複数指定できる。改行のみを入力するとモジュール ID の指定を終了する。

```
M-x qxt-spy-at-modules
ModuleId: m1
ModuleId: m2
ModuleId:
```

- qxt-list-spy
  スパイの一覧をカレントバッファに表示する。

```
M-x qxt-list-spy
```

を実行すると、以下のように表示する。Id: の後の数字をスパイポイント ID と呼ぶ。qxt-disable-spy、qxt-enable-spy、qxt-unspy-at-spypoint コマンド使用時に指定する。

```
** Subgoal spy Point **
(Id:1, enable) m:o1
(Id:2, enable) m:o2
** Rule spy point **
(Id:10, enable) m:r1
(Id:11, enable) m:r2
** Module spy point **
(Id:20, enable) m1
(Id:21, enable) m2
```

- qxt-unspy-at-subgoals
  サブゴールのスパイを解除する。Oterm: と聞いてくるので、オブジェクト項を指定する。サブゴールは複数指定できる。ModuleId: と Oterm: の両方で改行のみを入力するとサブゴールの指定を終了する。

```
M-x qxt-unspy-at-subgoals
ModuleId: m
Oterm: o
ModuleId:
Oterm:
```

- qxt-unspy-at-rules
  ルールのスパイを解除する。ModuleId: と聞いてくるので、モジュール ID を指定する。RuleId: と聞いてくるので、ルール ID を指定する。ルールは複数指定できる。ModuleId: と RuleId: の両方で改行のみを入力するとサブゴールの指定を終了する。

```
M-x qxt-unspy-at-rules
ModuleId m
RuleID: r
ModuleId
RuleID:
```

- qxt-unspy-at-rules
  モジュールのスパイを解除する。ModuleId: と聞いてくるので、モジュール ID を指定する。複数指定できる。改行のみを入力するとモジュールの指定を終了する。

```
M-x qxt-unspy-at-modules
ModuleId m
ModuleId
```

- qxt-unspy-at-spypoints
  スパイの種類を指定して、スパイを解除する。Indication: と聞いてくるので、subgoal、rule、module のいずれかを指定する。入力時にはコンプリーションが利用できる。SpyPointId: と聞いてくるので、スパイポイント ID を指定する。複数回指定できる。Indication: の入力時に改行のみを入力するとスパイポイントの設定を終了する。

```
M-x qxt-unspy-at-spypoints
Indication: rule
SpyPointID: 1
Indication:
```

- qxt-enable-all
  すべてのスパイをイネーブルする。

```
M-x qxt-enable-all
```

- qxt-enable-at-spypoints
  スパイの種類を指定して、スパイをイネーブルする。Indication: と聞いてくるので、subgoal、rule、module のいずれかを指定する。入力時にはコンプリーションが利用できる。SpyPointId: と聞いてくるので、スパイポイント ID を指定する。複数回指定できる。Indication: の入力時に改行のみを入力するとスパイポイントの設定を終了する。

```
M-x qxt-enable-at-spypoints
Indication: rule
SpyPointId : 1
Indication:
```

- qxt-disable-all
  すべてのスパイをディセーブルする。

```
M-x qxt-disable-all
```

- qxt-disable-at-spypoints
  スパイの種類を指定して、スパイをディセーブルする。Indication: と聞いてくるので、subgoal、rule、module のいずれかを指定する。入力時にはコンプリーションが利用できる。SpyPointId: と聞いてくるので、スパイポイント ID を指定する。複数回指定できる。Indication: の入力時に改行のみを入力するとスパイポイントの設定を終了する。

```
M-x qxt-disable-at-spypoints
Indication: rule
Spypoint: 1
Indication:
```

- qxt-save-spy
  スパイをセーブする。SaveSpyFile: と聞いてくるので、ファイル名を指定する。

```
M-x qxt-save-spy
SaveSpyFile: my-spy.qsp
```

- qxt-set-all-state
  スパイをロードする。LoadSpyFile: と聞いてくるので、ファイル名を指定する。コンプリーションが利用できる。

```
M-x qxt-set-all-state
LoadSpyFile: my-spy.qsp
```

- qxt-reset-trace
  トレース状態を初期状態にする。

```
M-x qxt-reset-trace
```

6.4.5 トレースセッションコマンド

問合せを実行中にトレースイベントが発生すると、トレースセッションになる。問合せの答がでると、データベースセッションに戻る。

- トレースイベント
  データベースセッションで、スパイを設定して M-x qxt-query を実行して、ゲートを通過した時トレースイベントになる。さらに、トレースセッションコマンドを実行して、ゲートを通過した時トレースイベントになる。以下のようにカレントバッファに表示される。

```
EventTag : Callsubgoal
NumberOfSolutions : 2
Tracing :  m:o[1/v]
SequenceID : 10
Children :
o1 <= o11, o12 ;;
o2 <= o21, o22 ;;
BelongingRule : non
LookupCycle : non
```

- qxt-set-gate-on-trace
  一つのトレースゲートを設定する。GateId: と聞いてくるので、call、exit、fail、lookup、bt、et、at、ic、update のいずれかを指定する。Switch: と聞いてくるので、on または off を指定する。入力にはコンプリーションが利用できる。

```
M-x qxt-set-gate-on-trace
GateId: call
```

- qxt-set-gate-all-on-trace
  すべてのトレースゲートを設定する。Call: Exit: Fail: Lookup: Bt: Et: At: Ic: Update: と順番に聞いてくるので、on または off を指定する。入力にはコンプリーションが利用できる。改行のみを入力すると off を指定したことになる。

```
M-x qxt-set-gate-all-on-trace
Call:
Exit:
Fail:
Lookup:
Bt:
Et:
At:
Ic:
Update:
```

- qxt-execute-by-trace-mode
  ステップノード、スパイノード、ノートレースノードを指定して実行する。StepNode: と聞いてくるので、ノード番号を指定する。複数回指定できる。改行のみを入力するとステップノードの指定が終了する。SpyNode: と聞いてくるので、ノード番号を指定する。複数回指定できる。改行のみを入力するとステップノードの指定が終了する。NotraceNode: と聞いてくるので、ノード番号を指定する。複数回指定できる。改行のみを入力するとステップノードの指定が終了する。

```
M-x qxt-execute-by-trace-mode
StepNode: 1
StepNode:
SpyNode: 2
SpyNode:
NotraceNode: 3
NotraceNode:
```

- qxt-execute-step
  ステップ実行する。

```
M-x qxt-execute-step
```

- qxt-execute-spy
  スパイになるまで実行する。

```
M-x qxt-execute-spy
```

- qxt-execute-notrace
  トレースしないで実行する。解が表示され、データベースセッションに戻る。

```
M-x qxt-execute-notrace
```

- qxt-list-spy-on-trace
  スパイの一覧をカレントバッファに表示する。

```
M-x qxt-list-spy-on-trace
```

- qxt-inspect
  トレースイベントを調べる。このコマンドを実行すると、インスペクトセッションになる。

```
M-x qxt-inspect
```

- qxt-abort-trace
  トレースをアボートする。質問はアボートとなり、データベースセッションに戻る。

```
M-x qxt-abort-trace
```

6.4.6 インスペクトセッションコマンド

トレースセッションで qxt-inspect コマンドを実行すると、インスペクトセッションになる。qxt-quit-inspect を実行すると、トレースセッションに戻る。

- qxt-inspect-assumption
  トレースイベント中の仮定部分を調べる。SolutionNumber: と聞いてくるので、解の番号を指定する。

```
M-x qxt-inspect-assumption
SolutionNumber: 1
```

- qxt-inspect-conclusion
  トレースイベント中の結論部分を調べる。SolutionNumber: と聞いてくるので、解の番号を指定する。

```
M-x qxt-inspect-conclusion
SolutionNumber: 1
```

- qxt-inspect-variable
  トレースイベント中の変数を調べる。SolutionNumber: と聞いてくるので、解の番号を指定する。VariableName: と聞いてくるので、変数名を指定する。

```
M-x qxt-inspect-variable
SolutionNumber: 1
VariableName: Var
```

- qxt-quit-inspect
  インスペクトを終了する。トレースセッションに戻る。

```
M-x qxt-quit-inspect
```

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.

[成果報告 92] ICOT. 平成3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. 1992.

[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 Memo. 1990.

[Tanaka 92] H. Tanaka. *Integrated System for Protein Information Processing. Proc. of the FGCS92*, Tokyo. June. 1992.

[Tojo 92] S. Tojo and H. Yasukawa. *Situated Inference of Temporal Information. Proc. of the FGCS92*, Tokyo. June. 1992.

[Yamamoto 91] N. Yamamoto. *TRIAL: a Legal Reasoning System (Extended Abstract). France-Japan Joint Workshop*, Renne, France. July. 1991.

[Yasukawa 92] H. Yasukawa, H. Tsuda, and K. Yokota. *Objects, Properties, and Modules in QUIXOTE. Proc. of the FGCS92*, Tokyo, June. 1992.

[Yokota 92] K. Yokota and H. Yasukawa. *Towards an Integrated Knowledge-Base Management System. Proc. of the FGCS92*, Tokyo, June. 1992.