Learning From Knowledge Systems

by

Craig Larman

B.Sc. Simon Fraser University, 1992

THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

in the School
of
Computing Science

Approval

NAME:	Craig Larman
DEGREE:	Master of Science
TITLE OF THESIS:	Learning from Knowledge Systems
Examining Committee:	
Chair:	Dr. Lou Hafer

Dr. Nick Cercone
Senior Supervisor

Dr. Veronica Dahl
Senior Supervisor

Dr. Ronald Harrop
External Examiner

Date Approved: June 19, 1995

Title of Thesis/Project/Extended Essay

Learning from Knowledge Systems

Author:
(signature)

Craig Larman
(name)

June 23/95
(date)

Abstract

This thesis describes a case-based reasoning (CBR) system, ISCN Student, which acquires its knowledge from a previously developed rule-based knowledge system, ISCN Expert. That is, ISCN Student is a second generation knowledge system that learns from a first generation one. ISCN Student has been shown to perform with the same competence as ISCN Expert once trained. The architecture for this solution is based upon the creation of a general purpose object-oriented CBR framework, written in Smalltalk, that has been specialized to develop ISCN Student, but which is applicable to other CBR problem domains.

ISCN is a notation used by geneticists to describe chromosome defects; the functional purpose of both ISCN Expert and Student is to interpret expressions in ISCN. To this end, several supporting paths of novel research have been pursued in addition to the above. First, a grammar and associated parser for ISCN were created. Second was the development of a visual manipulation system for displaying chromosome defects and introducing new abnormalities as cases to ISCN Student.

To Julie, Haley and Hannah,

Fred

and

my parents

Acknowledgments

My deepest heartfelt thanks to Drs. Nick Cercone and Veronica Dahl. So many opportunities have flowed from their support! Their memory, spirit and influence will be with me always. My gratitude and great thanks to Dr. Glen Cooper for suggesting the genesis of this work, for providing intellectual and material support, and for laying the strong foundation of ISCN Expert.

My sincere thanks to Dr. Ronald Harrop, who painstakingly examined this work and suggested many improvements.

Thanks also to the Science Council of British Columbia (in other words, the taxpayers), who provided the money to keep me going!

I owe a great debt to the developers of the language Smalltalk: Alan Kay, Adele Goldberg and Dan Ingalls, and to the creators of the Smalltalk parser generator, TGEN, Justin Graver and Hal Hildebrand.

Contents

Text Listings

List Of Figures

1. Introduction

A case-based reasoning (CBR) system, ISCN Student, whose input training cases are the output from a prior rule-based knowledge system, ISCN Expert, is described in this thesis. That is, ISCN Student is a second generation knowledge system that learns from a first generation one using case-based reasoning techniques. The problem domain is the interpretation of a notation used by cytogeneticists to represent chromosomal abnormalities, the International System for Human Cytogenetic Nomenclature (ISCN). The singular feature of ISCN Student is that it successfully interprets ISCN expressions, constructing a deep interpretation model, yet it has no inherent domain knowledge. ISCN Student acquires its competence by learning from ISCN Expert.

As well as the interpretation of ISCN expressions, alternate representations of both expressions and interpretations are investigated in this research. It uses a visual metaphor whereby one can view standard chromosome ideograms associated with an expression, and introduce new defects via direct visual manipulation. Visually modifying chromosome ideograms causes the underlying symbolic ISCN expression to be modified, which then causes the case-based reasoning component to attempt to construct a new interpretation model for the altered expression.

A further development in this work is the first definition of a generalized object-oriented framework model for case-based reasoning systems. As well, the first formal grammar specification of the ISCN language, which has been shown to be an LR(1) grammar, has been defined.

This work is of interest to several disparate groups. CBR researchers can use the general purpose object-oriented CBR framework as a common base for specialized systems, sharing common factored objects and behavior. This group may also find noteworthy aspects in the discussion of alternate input strategies using visual manipulation, and the potential for specialized matching and adaptation strategies which can be selectively introduced. Industry CBR developers can use the framework as a springboard to new systems, reducing reinvention. Researchers in automated learning and induction will find points of interest in the methods used to interpret input cases, and to automatically determine generalization concepts.

The thesis chapters cover the following material:

1. This introduction.

2. Background to this work, such as basic genetics and case-based reasoning (CBR) concepts. In this section an explanation of ISCN syntax is given, plus a review of related work.

3. A sample demonstration of ISCN Student at work. Screen captures of a session using ISCN Student illustrate the entire sequence of use. The effect on the semantic network of cases by using the system is illustrated.

4. Description of the ISCN grammar and how expressions are parsed. Abbreviated grammars for both the short and long form of ISCN are presented and illustrated with an example. The entire grammars are presented in Appendix A.

5. The system architecture from the point of view of process and object models. Data flow diagrams are used to describe the processes in ISCN Student and its relationship to ISCN Expert. An object model, or extented entity-relationship

diagram, based on OMT notation is given for the generic object-oriented framework for CBR systems.

6. Explanation of how the output interpretations from ISCN Expert are translated into cases in the CBR system. ISCN Expert creates a text file of interpretations; Student must read these and transform them into cases in a semantic network. This work includes the creation of generalized classes of concepts in the network.

7. Analysis of the process and logic used in the CBR *matching* phase. Given a new case represented as an ISCN expression, ISCN Student must find the closest existing cases in memory, and rank them from best to work match.

8. Analysis of the process and logic used in the CBR *adaptation* phase. Once a priority queue of closest matching cases is found, the adaptation phase must construct a new case from a synthesis of the existing cases. In this section a detailed description of the logic for this process is presented.

9. Overview of the visual manipulation subsystem: how the topology of abnormal chromosomes is computed, how the ideograms are drawn, and how visual changes initiate new CB reasoning processes.

10. Evaluation of the performance and correctness of ISCN Student. Student was measured with respect to the time required to interpret a new ISCN expression; the results are compared to the performance of ISCN Expert. Likewise, the correctness of Student's results are compared for new cases to that of ISCN Expert.

11. Future research possibilities and salient conclusions of this research.

1.1 Problem Statement

Create a CBR system that can interpret and encode the output of ISCN interpretations from ISCN Expert, representing them as cases in a CBR memory. When the CBR system is presented with new ISCN expressions create new interpretations that are as correct at those produced by ISCN Expert. As well, create a visual subsystem in the CBR system that displays the chromosomes and their defects, and which allows the introduction of new ISCN cases via visually manipulating the chromosome pictures.

1.2 Scope

ISCN Student has been limited to solve for a subset of sentences within ISCN grammar -- those dealing with terminal deletions and reciprocal translocations.

The visual manipulation system has been limited to display of chromosomes 1 through 5, and X and Y. This is because of excessive time delay in displaying any more chromosomes; the graphics are computationally intensive.

Case memory was limited to tests with twenty or less cases as input. Larger input sets are possible, but physical memory limitations created a physical constraint on the size of test cases.

2. Background

2.1 Genetic Terminology and Concepts

The genetic material of cells, which determines all physical characteristics and functions, is represented in *chromosomes*, which are composed of tightly packed DNA molecules. In each cell of a normal human is found an identical set of 46 chromosomes, which are a configuration of 23 pairs: the sex pair ("XX" for females, "XY" for males) and 22 pairs of *autosomes*, numbered 1 through 22. One member of each pair is inherited from the father, the other from the mother.

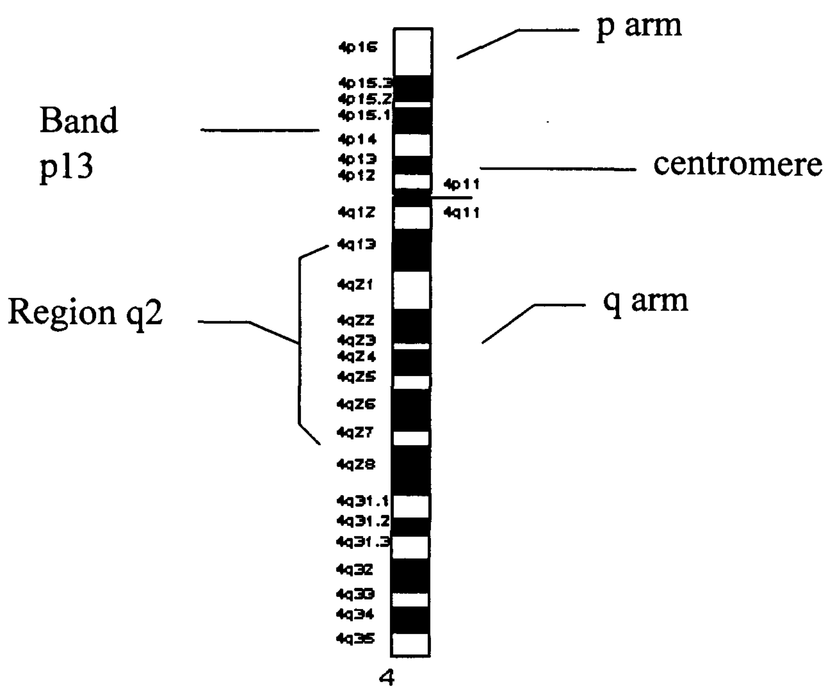


Figure 2.1 The structure of chromosome 4

A chromosome has a sequence of contiguous sections that appear in alternating light and dark bands when stained using various chemical techniques. Each chromosome has a characteristic constriction called the *centromere*, which divides the chromosome into a short (petite) 'p' *arm* and a longer 'q' arm. The arms are segmented into major sections called *regions*, that are numbered in increasing order from the centromere outward. For example, regions 4p1, 4p2, ... Further, regions may be subdivided into sections called *bands*, such as 4p11, 4p12, 4p21, 4p22, ... (Figure 2.1).

Chromosomal abnormalities involve deviations with respect to the number of chromosomes (*numeric abnormalities*) or their structure (*structural abnormalities*). They may be either congenital, such as inherited from the parents, or acquired, such as due to a cancer. Physical abnormalities usually result from chromosomal abnormalities. For example, a numeric abnormality in which there are 3 rather than 2 chromosome number 21s results in Down Syndrome. A structural abnormality in which a terminal portion of chromosome number 5 is deleted (missing) may result in Cri du chat syndrome.

Cytogeneticists specialize in the investigation of chromosome abnormalities. For inspection, a sample set of chromosomes from a cell is isolated and stained to produce a fluorescent banding pattern. A picture is taken and the individual chromosomes are identified, enlarged and grouped into a *karyotype* for visual inspection. Chromosome defects can then be identified and recorded. The symbolic short-hand notation used to describe the karyotype and the observed abnormalities is called ISCN.

2.1.1 Structural Chromosomal Abnormalities

There are many categories of structural abnormalities, this discussion will use only two, in order to simplify examples.

The first is a *deletion* (or *terminal deletion*), in which the end portion of a chromosome arm is broken off and lost. The break can be identified by the chromosome number and band at which the break occurred.

The second is a *translocation* (or *reciprocal translocation*), in which there is an exchange of chromosomal segments between two chromosomes. The break can be identified by the two chromosomes, and for each, the band at which the segment broke.

2.2 ISCN Expressions

Cytogeneticists describe chromosomal defects using a standard notation, the International System for Human Cytogenetic Nomenclature (ISCN). This notation describes the:

- number of chromosomes
- sex chromosomes
- numerical abnormalities
- structural abnormalities

For example:

Normal Female - 46 chromosomes with the XX sex chromosomes

46,xx

Numeric Abnormality - Extra Chromosome 21 (female with possible Down Syndrome)

47,xx,+21

Numeric Abnormality - Missing Chromosome 15

45,xx,-15

Structural Abnormality - Terminal Deletion at Band p14 on Chromosome 5. (Figure 2.2)

46,xx,del(5)(p14)

Structural Abnormality - Translocation between chromosomes 2 and 3, with breakage's at 2q31 and 3p21. (Figure 2.3)

46,xx,t(2;3)(q31;p21)

Combination of Abnormalities

47,xx,+2,t(2;3) (q31;p21),del(5)(p14)

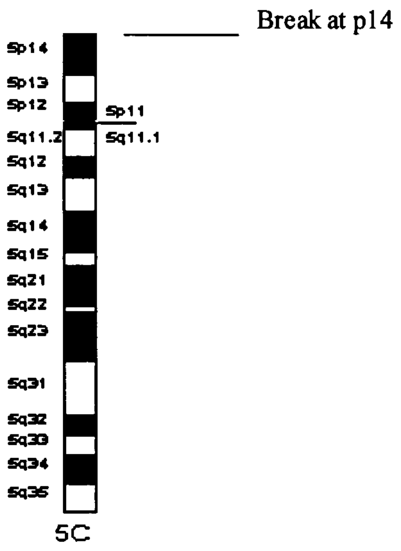


Figure 2.2 Chromosome 5 for 46,xx,del(5)(p14)

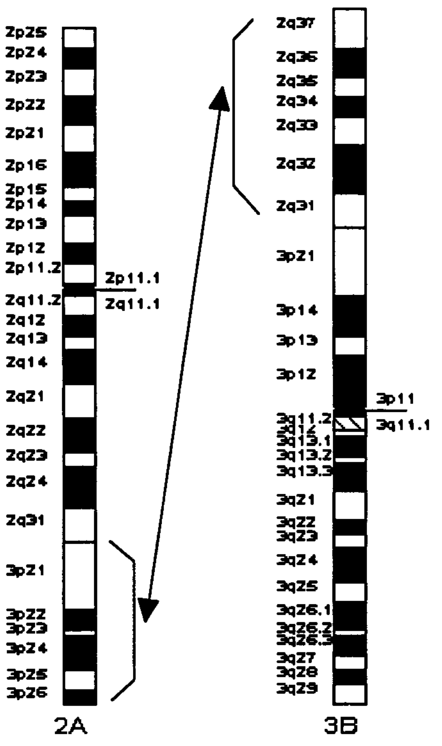


Figure 2.3 **Chromosomes 2 and 3 for 46,xx,t(2;3)(q31;p21)**

2.2.1 ISCN Long Form Expressions

ISCN defines a detailed notation for structurally modified chromosomes. A double colon (::) is used to indicate break and reunion, an → is used to indicate a from-to range, the end of a chromosome is signified as *qter* or *pter*, and the centromere is denoted by *cen*.

For example:

ISCN Short Form	ISCN Long Form
46,xx,del(1)(q21)	*46,xx,del(1)(pter→q21)*
46,xx,del(1)(q21q31)	*46,xx,del(1)(pter→q21::q31→qter)*

2.3 From Expert to Student

ISCN Expert, the prior first generation knowledge system, uses explicit domain knowledge represented in the form of Prolog rules to interpret ISCN expressions (Cooper 1990). This hard-won knowledge was gathered from intensive study of related literature (Harden 1985) and extensive interviews with a domain expert (Friedman 1986). ISCN Expert has achieved an expert level of performance and can interpret a very substantial subset of ISCN expressions, including those that contain complex and subtle issues of interpretation. ISCN Expert produces as output a detailed textual interpretation of the ISCN expression.

The genesis of ISCN Student, a second generation system, was to extend ISCN Expert in some interesting or useful way; for example, to increase the knowledge base or investigate the visualization of chromosomes. An additional goal (of the author) was to ensure the emphasis of the research fell within the Computing Science domain, rather than being a exercise in pure genetics knowledge acquisition. Reflection on these goals, in conjunction with parallel research into models of medical knowledge systems, and the generic use of case-based reasoning systems, led to an inspiration: it would be both possible and intriguing to explore the creation a second generation case-based reasoning system that would learn, via the ISCN Expert interpretation cases, to also interpret ISCN expressions. This second generation system, ISCN Student, also afforded the opportunity to investigate other interesting research questions, such as how to model a visual manipulation system for chromosomal defects, and how to create an object-oriented model for CBR systems.

2.4 Case-Based Reasoning

Case-based reasoning (CBR) draws inspiration from a cognitive model emphasizing memory, recollection and adaptation in problem solving and interpretation. The seminal work in the field is from Roger Schank and his researchers in the Yale AI lab (Schank 1982, Kolodner 1986). The CBR model is based on reasoning in new situations by remembering previous similar *cases* of problems and solutions and *adapting* the old *solutions* to fit the new case situation.

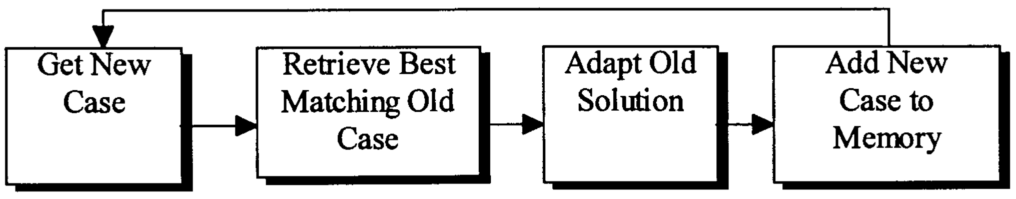


Figure 2.4 A Basic Case-based Reasoning Cycle

A CBR system has a dynamic memory that records old cases and their solutions. Reasoning about a new input problem case involves finding the most similar matching old case or cases and adapting its solution. The new case and its solution are then added to memory, and in this sense the system can be said to dynamically learn (Figure 2.4). CBR systems can also acquire domain knowledge by reading new sets of completed cases (with their solutions). A CBR memory starts as an empty shell without cases; new ones may be added via reasoning and adaptation, or via input of completed cases. As case memory grows performance improves, due to the system's larger set of cases to match against new ones. The likelihood of matching to an existing case needing little or no adaptation increases as the case memory grows.

2.4.1 CBR Components

A CBR system is composed of:

- A **memory** of cases and supporting knowledge structures.

  The usual representation of memory is a set of **MOPs** (Memory Organization Packages). A MOP is a frame-like object that is the basic unit of memory (Schank 1982). MOPs are organized in a graph structure to create a MOP-based memory which incorporates generalization-specialization hierarchies, multiple inheritance and composition (or arbitrary linkage) relationships. Their use differs from classic frame or script systems (Minsky 1975; Schank & Abelson 1977) in their application to dynamically changing knowledge bases that are altered during CBR reasoning. A MOP is either an *abstraction* or *instance*: the former representing a class-type object, the latter a specific instance of the abstraction.

- A **pattern matcher** for matching new cases to old.
- An **adapter** which adapts the solutions of old cases to new ones.

A human user (or other external system) inputs a new case to the CBR system. It matches the new cases to existing ones and uses the best matches to adapt a new solution for the new case, which is then added to memory (Figure 2.5).

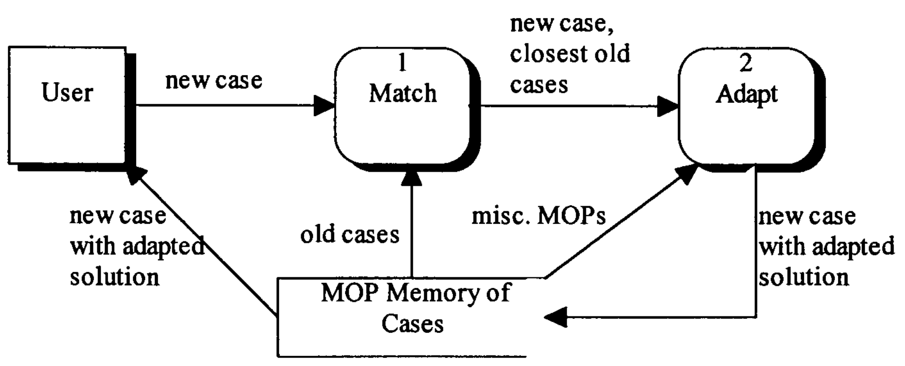


Figure 2.5 Data Flow Diagram of a Generic CBR System

2.4.2 CBR Advantages

- Simple model; quick easy development.

A virtue of CBR in the context of developing real-world applications is its simplicity; conceptually the model is easy to grasp, and there is evidence to show that system development is quicker and easier compared to other AI programming techniques, such as rule or model-based systems (Goodman 1989, Koton 1988).

- Easy knowledge acquisition; access to large volumes of domain knowledge in the form of cases.

Knowledge engineering, either in terms of identification of heuristic rules or of first principles causal models, is painstaking work and prone to error. Plus it requires the services of a very highly skilled individual. CBR systems, in contrast, have available to them the history of problems and solutions, which are relatively easier to both collect and record. Businesses often have large quantities of computer readable past completed cases.

- Speed; avoiding the effort to recreate solutions from scratch.

CBR systems have been shown to outperform other approaches, such as model-based programs (Koton 1988), because of the relatively simply computational model. As the size of case memory grows there is a dynamic tension of trade-offs with respect to performance: potential for slowing the performance because of searching a larger memory, versus speeding time to solution by finding an existing case in the large set of cases that very closely matches the new problem. Research into this problem, and the quick, efficient retrieval of appropriate cases is termed the *indexing problem.* As case memory grows, speed to matches can be improved by the selection of a suitable set of indices that provide keys for rapid convergence to appropriate old cases. The choice of these indices in non-trivial. Alternative approaches to rapid search are now being explored that involve concurrent threads of execution in multiprocessor computers.

- Improved problem solving with use.

CBR systems have the capacity to improve performance, in terms of solving more problems and more quickly, with use; in other words, they learn. As new cases and their solution are added to memory, the CBR case set grows and the likelihood increases of closely matching a new case to a similar old case (with a relevant solution requiring little or no adaptation). In contrast, rule-based systems do not improve with use; only by the *knowledge engineering* process of manually adjusting or adding rules to the rule base does it become a better problem solver. In practice, fielded rule-based systems tend to suffer maintenance problems because the users don't possess the expertise to update the rule base. Likewise, neural networks require a discrete training session separate from their use in order to be modified. The ability of CBR systems to automatically improve with use makes them very attractive from a maintenance perspective. Moreover, users may fine-tune the learning process by vetting and adjusting newly adapted solutions to input cases before they are added to memory.

The ability of CBR systems to learn from past cases and to match on inexact inputs is analogous to the advantages that neural networks possess. Yet CBR offers an improvement over nets in its capacity to handle higher-level cognitive processing and symbolic representations; nets aren't yet good at this. Research in connectionist models that attempt to represent symbolic CBR memories may eventually eliminate their disadvantage.

- CBR can handle inexact inputs gracefully.

CBR is potentially robust in handling noise and uncertainty in the input data; the matching phase can allow for partial pattern matching on novel cases. The developer of the matching algorithms can define to a greater or lesser extent the number of attributes that are required to match, and the strength of attribute value matching. For example, attribute value matching can be based on strict equality, or commonality within a range of values.

- Solutions in domains where algorithms, rules or causal models are not developed or understood.

CBR systems need only problem cases and their solutions, and a means to adapt old solutions to meet variations in new cases.

2.5 Related Work

2.5.1 Related ISCN Interpretation Work

Computerized interpretation of ISCN was first described by Dr. Jan Friedman (Friedman 1986), a geneticist who attempted the construction of an interpretation system (with the aid of professional business programmers) in the language COBOL, without the benefit of knowledge or techniques from the domains of language theory or Artificial Intelligence, such as parsing, semantic representations, rule-based knowledge representation, etc.

The problem was revisited in the latter 1980's by Dr. Glen Cooper, at the request of Friedman (Cooper 1990). Cooper applied logic programming techniques, such as explicit rule-based knowledge representation, using the language PROLOG, in order to develop ISCN Expert - a very robust ISCN interpreter.

The architecture and solution strategy of ISCN Student is radically different from Friedman's or Cooper's. Friedman's system, as one would expect of a program written in COBOL without the benefit of AI methods, lacked representation richness and sophistication of reasoning methods. All information was stored in global variables or records of primitive data types; there was no representation for frame-like semantically rich concepts, or a semantic network relating concepts. Cooper's system did include richer symbolic structures for different aspects of the expressions interpretations, stored as Prolog clauses, but their potential relationships were not expressed in a network and their was no use of specialization hierarchies to factor and share representation.

The reasoning and control in Friedman's system suffered from the classical problem of mixing knowledge and control in a procedural programming language - it was not at all clear what the rules of interpretation were, meta-reasoning was impossible, and modification was unwieldy. Cooper's system made a vast improvement by using a rule-based solution in Prolog in which clauses matched and fired using Prolog's built-in backward chaining inference mechanism. Its main weaknesses were a computationally intensive exhaustive search which relied heavily on blind backtracking to match the rules of interpretation to the ISCN expression, and lack of meta-logically reasoning or learning to improve performance.

ISCN Student builds primarily on Cooper's work in ISCN Expert, as the input learning cases for Student are the output from Expert. However, Student is more limited in its scope of expressions than Expert, as Student's emphasis is on the proof of concept for a CBR learning architecture, whereas Expert's emphasis is on a very thorough interpretation of most ISCN expressions, suitable for acceptance by cytogenetic experts. Therefore Student was constrained to expressions involving numeric abnormalities and deletion and translocation genetic defects.

Besides performing an identical task as Expert, that is, interpretation of expressions, Student has a variety of functions and methods which distinguish it. Undoubtedly Student's singular novel feature is its ability to learn problem solving skills from existing knowledge systems. Further, Student's initial knowledge base is automatically induced from a set of learning cases, where Expert's was hand-crafted via a human knowledge engineering effort. Student includes a learning function that modifies its performance in new problems, learning from prior solutions. While Expert's knowledge base and reasoning don't change over time, Student solves new problems drawing on previous solutions. In contrast to Expert, Student uses a semantically rich network of related concepts, associated by specialization and composition relationships. From this, reasoning based on inheritance and aggregation, such as inheritance of abnormalities, was extensively exploited. While Expert's reasoning was based directly on Prolog's inference mechanism, Student used a CBR match-and-adapt inference engine which, it is claimed, more accurately models the human reasoning process in this domain, and which provided the basis for meta-level programming capabilities such as dynamic criticism and tracing. Another significant advancement made by Student is the visual representation and manipulation possible; Expert allowed only symbolic character string inputs and outputs.

It is not established how well Student's performance would scale up to a broader range of expressions and cases from Expert, but it would certainly require enhancement to the case input and adaptation subsystems. Representation changes would not be necessary, as the frame-like structures (MOPs) used in ISCN Student are automatically derived from the input cases using an abnormality-independent algorithm.

2.5.2 Related Genetics Work

Dr. David Searles (Searles 1993) has done research into defining a grammar for DNA sequences, based on definite clause grammars implemented in Prolog. From this grammar, parsing and reasoning processes are being explored. Note that the DNA level of genetic representation is lower than the chromosome level; chromosomes being composed of tightly wound coils of DNA. Besides very different aims to ISCN Student, Searles' work emphasizes the microcosmic level to ISCN Student's macrocosmic level in the world of genetics. As with ISCN Expert, this research was limited to symbolic character string representations, in contrast to ISCN Student's visualization capabilities.

2.5.3 Related CBR Work

ISCN Student relies on the case-based reasoning (CBR) model of problem solving (Schank 1982), which is discussed in greater detail in section 2.4. Kolodner lays out the state of the art in (Kolodner 1993). She provides an in-depth look at representation and reasoning issues through many case studies, but focuses on the semantic level, rather than the architectural or implementation level. Where implementation is discussed, such as in some of Kolodner's cases, or Schank's survey of CBR implementation techniques (Schank 1989), the solutions are consistently function-oriented, usually written in Lisp, and without an organizing systems metaphor. In contrast to existing CBR literature's emphasis on semantic level issues, albeit their preeminent importance, a contribution embodied in ISCN Student is the definition of an architectural-level solution: an object-oriented framework for a generalized CBR system which can be specialized for domain dependent CBR applications.

What further distinguishes ISCN Student from existing CBR work is the source of its input cases used for learning - an existing knowledge system. Computer systems that perform automated knowledge acquisition from other computer systems are presumably uncommon, because a literature search yielded no references of related work. The closest work in this area is CASEY (Koton 1988), a CBR system for heart failure diagnosis that is built on top of a previous model-based system, the Heart Failure Program (Long 1987). If a new case presented to CASEY can't be solved, it passes it on to HFP, which returns its results for CASEY to use in subsequent cases. CASEY's training cases do not, however, derive from HFP outputs.

2.5.4 Related Visualization Work

Recent work in scientific visualization is surveyed in (Hagen 1993) and (Nielson 1990). Interestingly, throughout the descriptions the emphasis is on passive display of results, rather than on visually manipulating the images to affect the underlying models or objects they represent, although this idea is not new - most recently explored in virtual reality research. A specific (and apparently not widely exploited goal) of the visualization subsystem of ISCN Student was to link the visualization back to the underlying model so that there is a two-way linkage.

Along this vein, Student is the only CBR system (at least as far as literature searches yield) to accept case input via a direct visual manipulation metaphor; all existing CBR systems receive their cases in a symbolic character string format.

In the literature on biomedical computer systems for genetics, there is no report on the visualization of chromosome ideograms. ISCN Student contributes to this area by defining a representation to store chromosome ideogram graphical information, and a means to (relatively) efficiently display ideograms.

3. Example: ISCN Student at Work

A sample session using ISCN Student will illustrate its central features.

On startup, the main window is displayed which shows a list of existing cases in memory, and various action buttons (Figure 3.1).

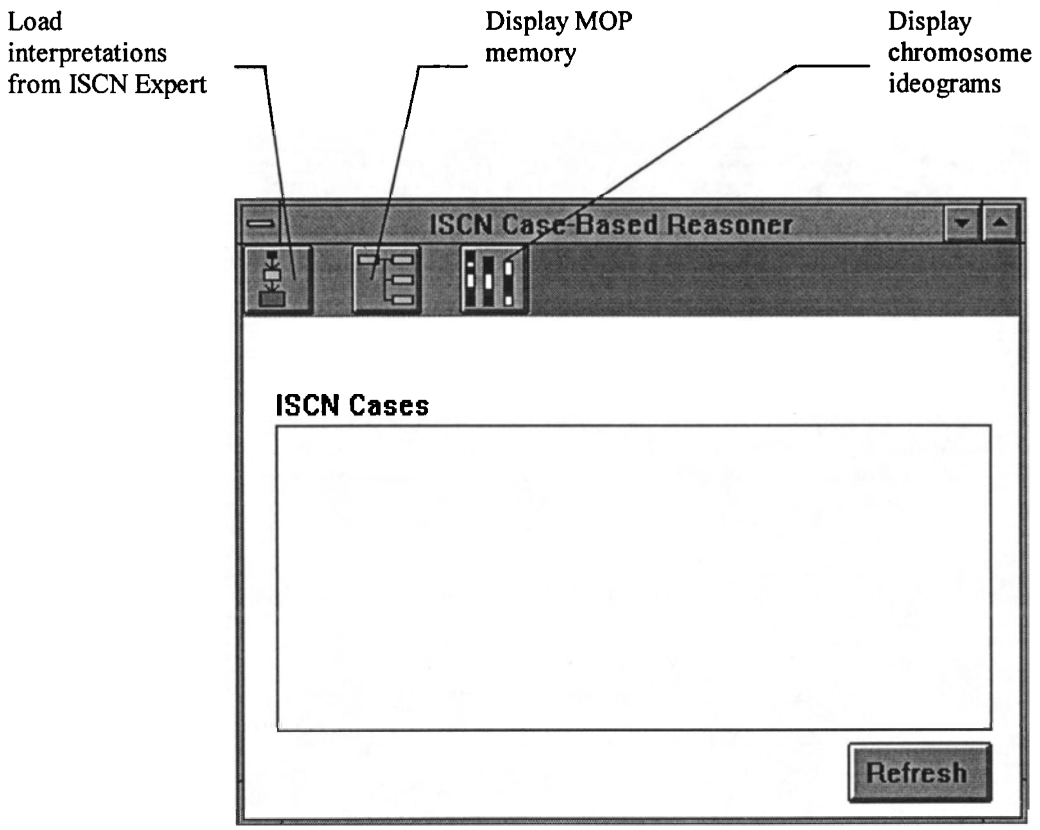
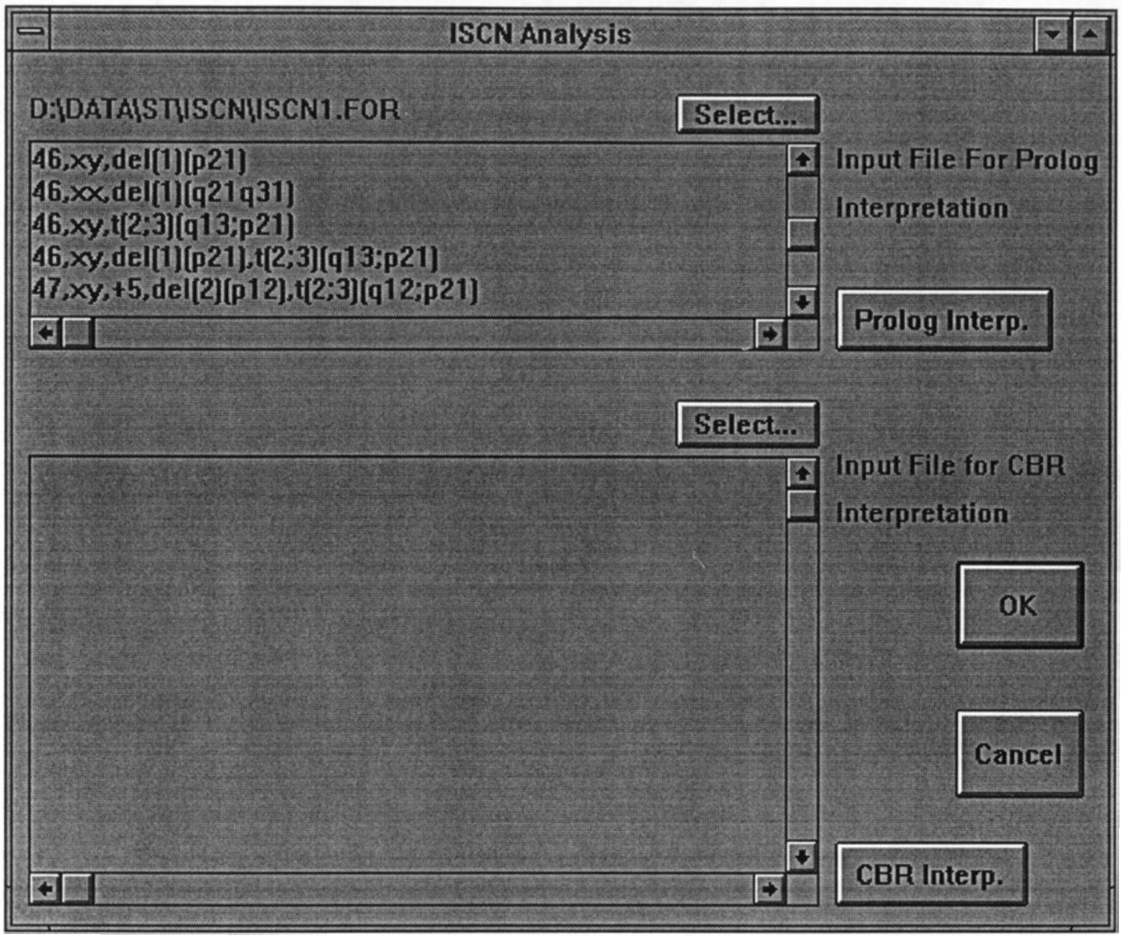

Load interpretations from ISCN Expert
Display MOP memory
Display chromosome ideograms
ISCN Case-Based Reasoner
ISCN Cases
Refresh

Figure 3.1 Opening ISCN Student Window

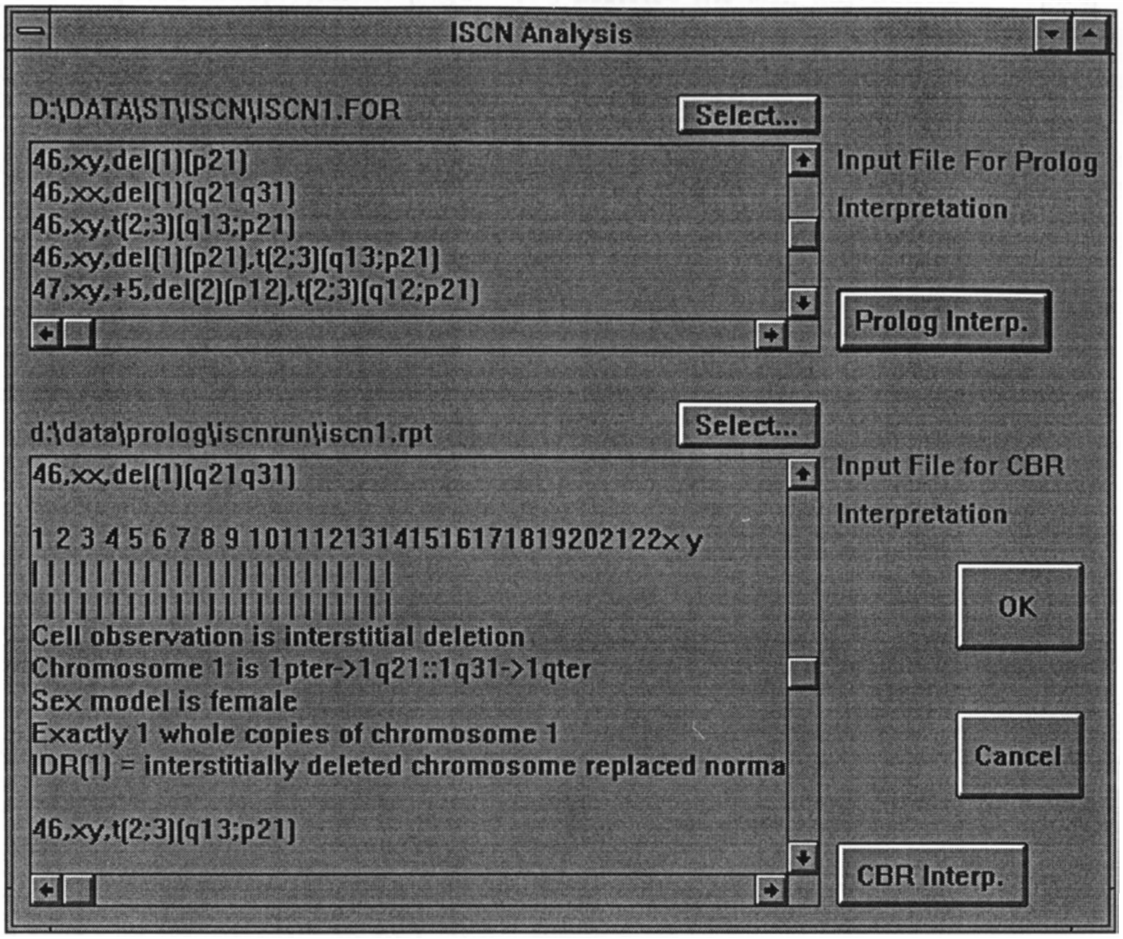
When the ISCN Analysis window is opened, one can enter ISCN expressions directly, or select them from a text file (Figure 3.2).


ISCN Analysis
Select...
Input File For Prolog Interpretation
Prolog Interp.
Select...
Input File for CBR Interpretation
OK
Cancel
CBR Interp.

```
D:\DATA\ST\ISCN\ISCN1.FOR
46,xy,del(1)(p21)
46,xx,del(1)(q21q31)
46,xy,t(2;3)(q13;p21)
46,xy,del(1)(p21),t(2;3)(q13;p21)
47,xy,+5,del(2)(p12),t(2;3)(q12;p21)
```

Figure 3.2 Translation Window to invoke ISCN Expert and Load Interpretations

The "Prolog Interp." button causes the ISCN expressions to be interpreted to a text output format using ISCN Expert (Figure 3.4). Once completed, the interpretations can be loaded into CBR memory via the "CBR Interp." button (Figure 3.3). In this figure we see the ISCN Expert output for the expression 46,xy,del(1)(q21q31), which will be input (along with the other interpretations) to the CBR load phase that translates them into MOP memory cases.


ISCN Analysis
Select...
Input File For Prolog Interpretation
Prolog Interp.
Select...
Input File for CBR Interpretation
OK
Cancel
CBR Interp.

D:\DATA\ST\ISCN\ISCN1.FOR

```
46,xy,del(1)(p21)
46,xx,del(1)(q21q31)
46,xy,t(2;3)(q13;p21)
46,xy,del(1)(p21),t(2;3)(q13;p21)
47,xy,+5,del(2)(p12),t(2;3)(q12;p21)
```

d:\data\prolog\iscnrun\iscn1.rpt

```
46,xx,del(1)(q21q31)

1 2 3 4 5 6 7 8 9 10111213141516171819202122x y
| | | | | | | | | | | | | | | | | | | | | | |
 | | | | | | | | | | | | | | | | | | | | | | |
Cell observation is interstitial deletion
Chromosome 1 is 1pter->1q21::1q31->1qter
Sex model is female
Exactly 1 whole copies of chromosome 1
IDR(1) = interstitially deleted chromosome replaced norma

46,xy,t(2;3)(q13;p21)
```

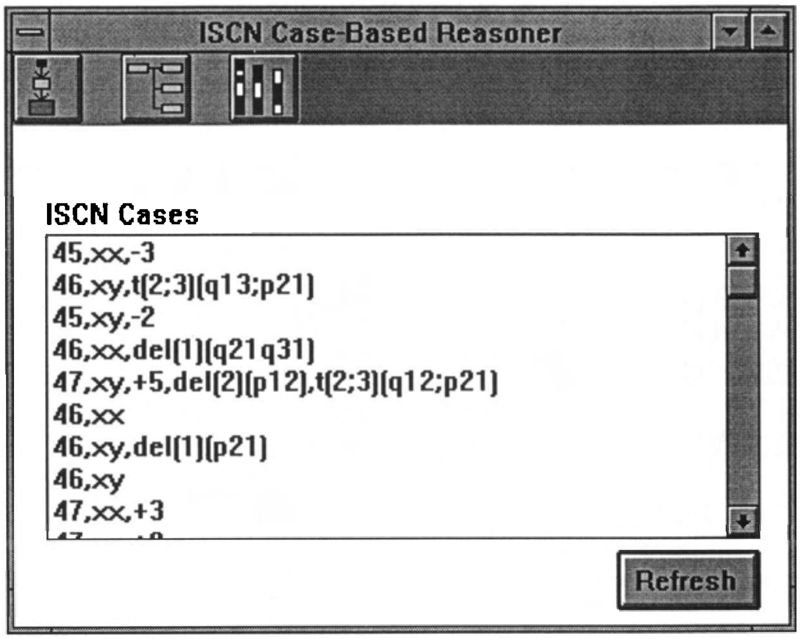
Figure 3.3 ISCN Expert's Translation Results are Ready for CBR Interpretation

ISCN Expert's interpretations (Figure 3.4) describe the sex, any abnormal copies, the detailed structure of abnormal chromosomes, and the nature of the abnormalities.

```
47,xy,+2

1 2 3 4 5 6 7 8 9 10111213141516171819202122x y
| | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | |
  |
Sex model is male
Exactly 3 whole copies of chromosome 2
XNX(2) = extra whole normal chrom (trisomy/xxy/xyy)



47,xy,+5,del(2)(p12),t(2;3)(q12;p21)

1 2 3 4 5 6 7 8 9 10111213141516171819202122x y
|   | | | | | | | | | | | | | | | | | | | | | |
|     | | | | | | | | | | | | | | | | | | | |
        |
Cell observation is reciprocal translocation
Cell observation is terminal deletion
Chromosome 2 is 2pter->2q12::3p21->3pter
Chromosome 3 is 2qter->2q12::3p21->3qter
Chromosome 2 is 2p12->2qter
Sex model is male
Exactly 1 whole copies of chromosome 3
Exactly 0 whole copies of chromosome 2
Exactly 3 whole copies of chromosome 5
RTB(2,3) = balanced carrier of reciprocal translocation
TDR(2) = terminally deleted chromosome replaced normal
XNX(5) = extra whole normal chrom (trisomy/xxy/xyy)
```

Figure 3.4 **Sample ISCN Expert Interpretation Cases Output**

Once the ISCN Expert interpretations have been loaded, the ISCN Student main window can be refreshed to show the new list of cases loaded in MOP memory (Figure 3.5).



```
45,xx,-3
46,xy,t(2;3)(q13;p21)
45,xy,-2
46,xx,del(1)(q21q31)
47,xy,+5,del(2)(p12),t(2;3)(q12;p21)
46,xx
46,xy,del(1)(p21)
46,xy
47,xx,+3
```

Figure 3.5 **ISCN Student's Main Window after Loading Interpretation Cases**

The "Memory Display" button can be used to display a graph of all MOPs in memory - a subject which will be discussed in greater detail later. The graph shown here (a directed acyclic graph) illustrates the cases that have been loaded into memory from ISCN Expert. Generalizations are to the left, specialization's to the right. Note that generalized abstraction cases have been automatically created, such as 47,+<num>, from instances such as 47,xx,+3 (Figure 3.6).

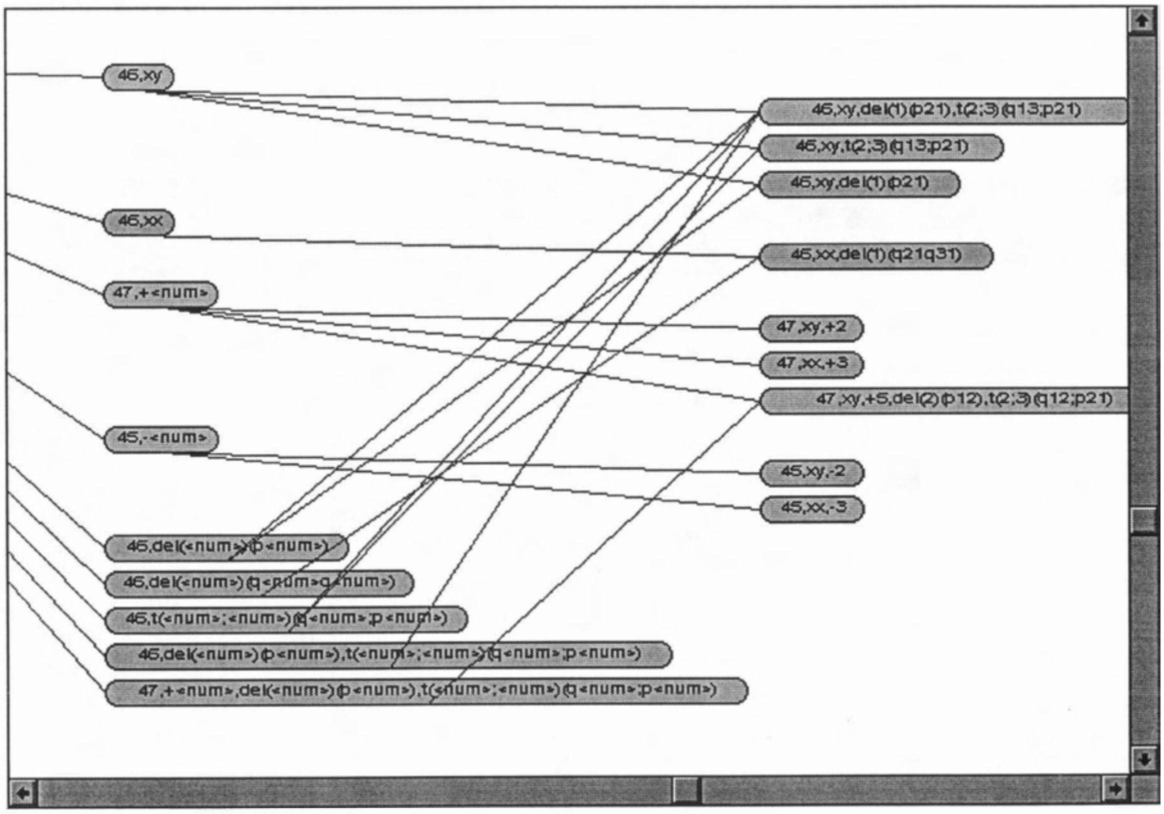
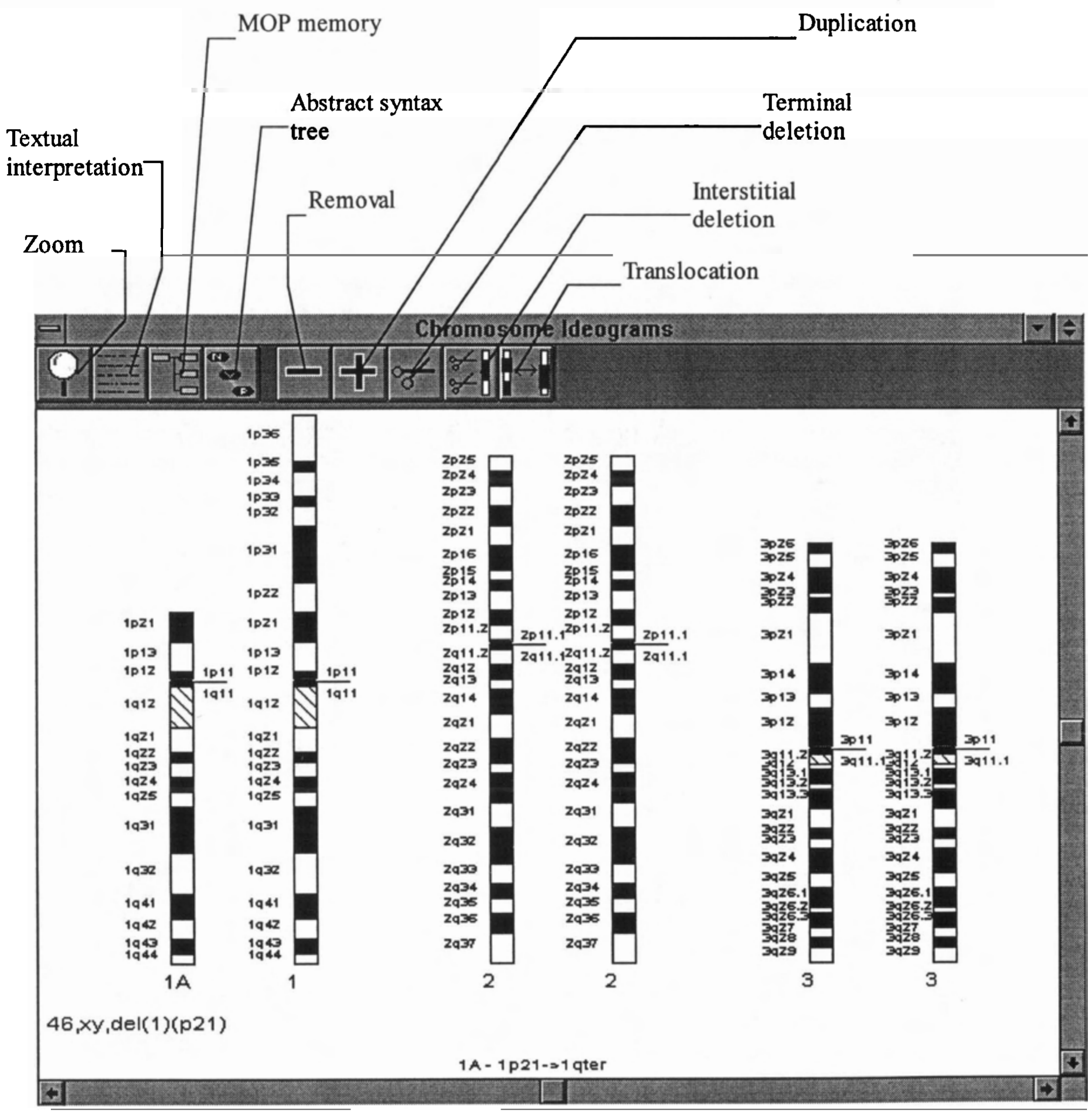

46,xy
46,xx
47,+<num>
45,-<num>
46,del(<num>)(p<num>)
46,del(<num>)(q<num>q<num>)
46,t(<num>;<num>)(q<num>;p<num>)
46,del(<num>)(p<num>),t(<num>;<num>)(q<num>;p<num>)
47,+<num>,del(<num>)(p<num>),t(<num>;<num>)(q<num>;p<num>)
46,xy,del(1)(p21),t(2;3)(q13;p21)
46,xy,t(2;3)(q13;p21)
46,xy,del(1)(p21)
46,xx,del(1)(q21q31)
47,xy,+2
47,xx,+3
47,xy,+5,del(2)(p12),t(2;3)(q12;p21)
45,xy,-2
45,xx,-3

Figure 3.6 **MOP Specialization Hierarchy for Interpretation Cases**

From the main window, one can also choose an existing case, and display the chromosome ideograms for it (Figure 3.7). In this instance, the ideograms for 46,xy,del(1)(p21) are displayed. Note that the abnormal chromosome (1) is shown with its correct structure; a terminal deletion after 1p21. It is coded as "1A", and the lower centre of the display shows the *ISCN long form* expression of the structure (1p21→1qter).

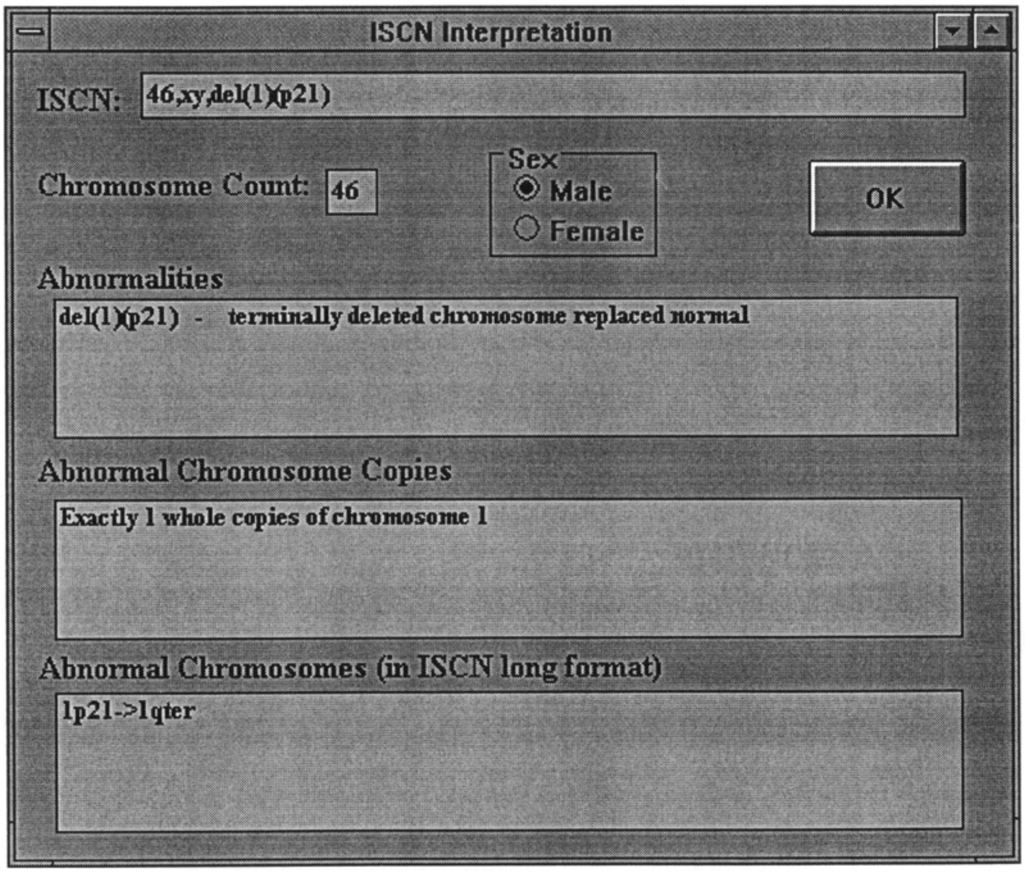


46,xy,del(1)(p21)

1A - 1p21->1qter

Figure 3.7 **Subset of Chromosome Ideograms for 46,xy,del(1)(p21). Note Ideogram 1A.**

From the chromosome ideogram window one can display a textual interpretation of the expression (Figure 3.8). This interpretation is stored in MOP memory via associative relationships with the case. Note that the sex, total number of chromosomes, and abnormality expressions, for example *del(1)(p21),* have been isolated, and a description attached. Likewise, abnormal chromosome numbers and ISCN long form expressions are expressed for all aberrations. This interpretation is evidence of a deeper semantic understanding of the surface ISCN expression, and is constructed from associations maintained in MOP memory.



ISCN: 46,xy,del(1)(p21)

Chromosome Count: 46

Sex: [✓ Male] Female

Abnormalities

del(1)(p21) - terminally deleted chromosome replaced normal

Abnormal Chromosome Copies

Exactly 1 whole copies of chromosome 1

Abnormal Chromosomes (in ISCN long format)

1p21->1qter

Figure 3.8 Textual Interpretation for 46,xy,del(1)(p21)

An abstract syntax tree (AST) may be generated for expressions, from the ideogram window (Figure 3.9). An LR(1) parser has been constructed for the ISCN grammar, which is used to construct the trees. An exhaustive derivation tree may also be generated, but is excluded from this view because of its verbosity. The AST view is not directly useful, but the underlying AST and derivation trees are used extensively during the CBR matching and adaptation phases.

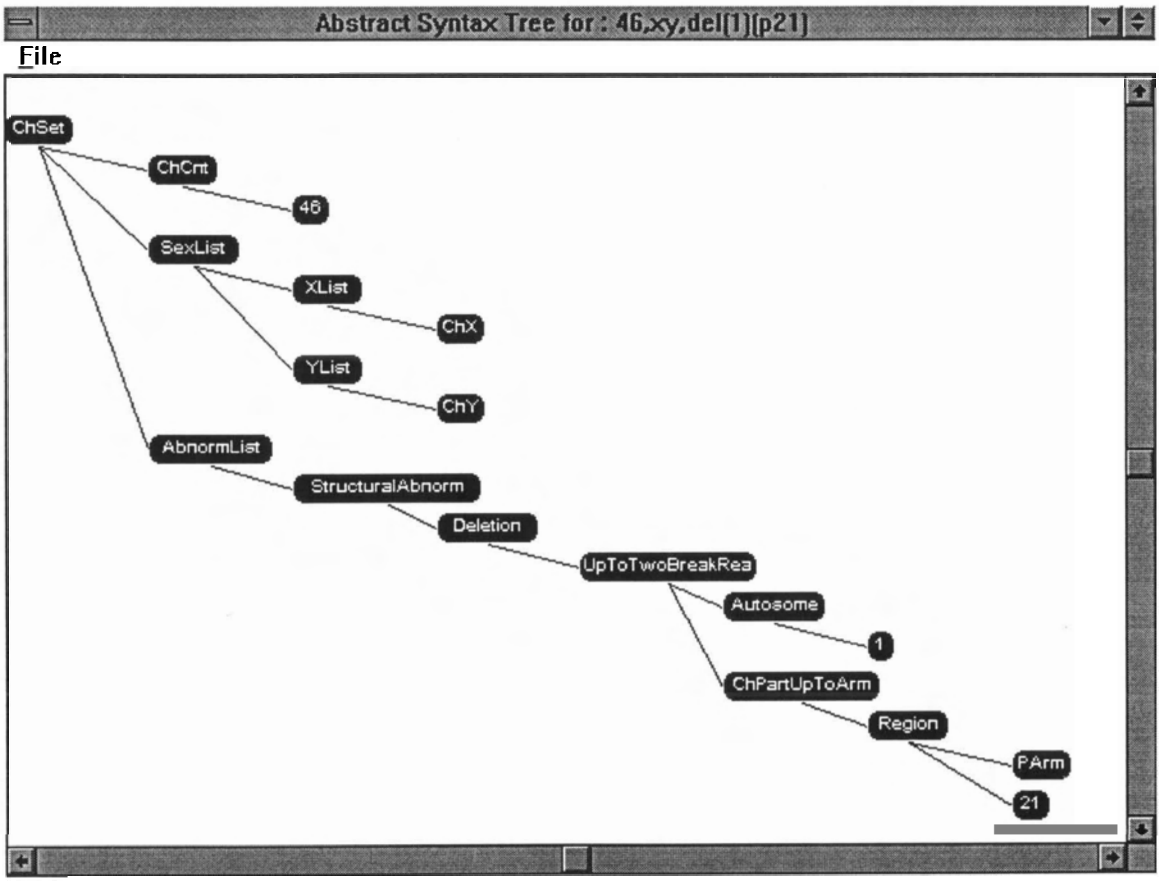


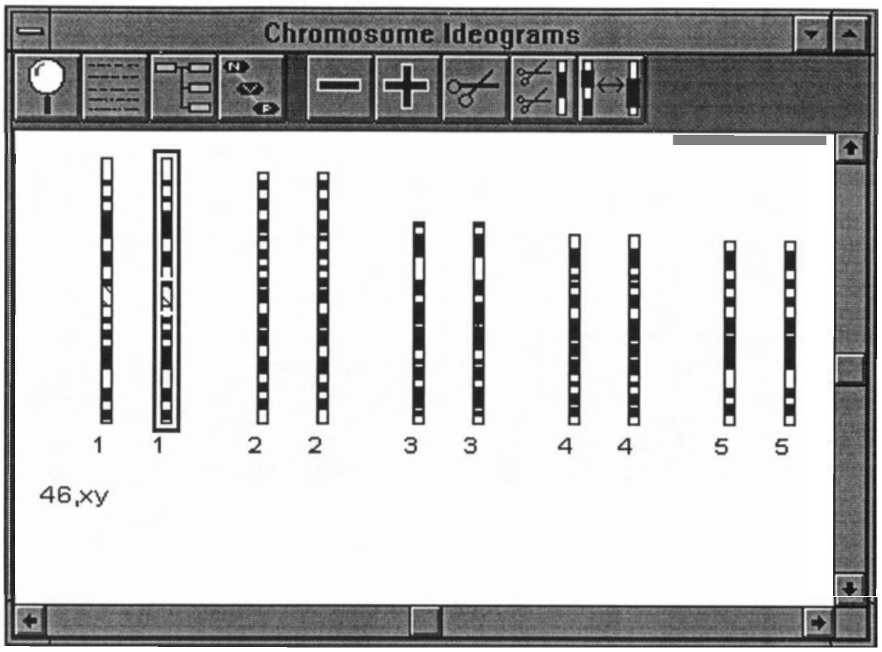
Figure 3.9 **Abstract Syntax Tree for 46,xy,del(1)(p21)**

An interesting feature of ISCN Student is the use of visual manipulation to introduce new ISCN expressions to the CBR engine for potential interpretation (Figure 3.10). One can add or remove chromosomes, or add structural abnormalities using a visual metaphor. The singular feature of this utility is that these changes only affect the surface syntax of the ISCN expression, not the underlying chromosome model or case represented in MOP memory. For example, visually choosing a chromosome and requesting to duplicate it (e.g., choosing to duplicate chromosome 1) causes only the surface expression to change with the addition of a "*+1*" to the ISCN expression. The visual manipulator subsystem, and indeed the entire CBR system, has no deep knowledge of chromosome models or ISCN semantics, and thus is incapable of affecting a deep change. The manipulator subsystem has knowledge of surface syntax only, knowing that when a chromosome duplication is requested, a "*+?*" can be added to the expression.

How then, is the underlying model or case updated to reflect the visual manipulations? Each time a visual modification causes the ISCN expression to be modified, the new resultant expression is treated as a *new case* for the CBR engine to interpret. The new expression is matched to existing cases, and a deep semantic model (a *solution*, in this domain) is constructed via adaptation from existing models (recall section 2.4).

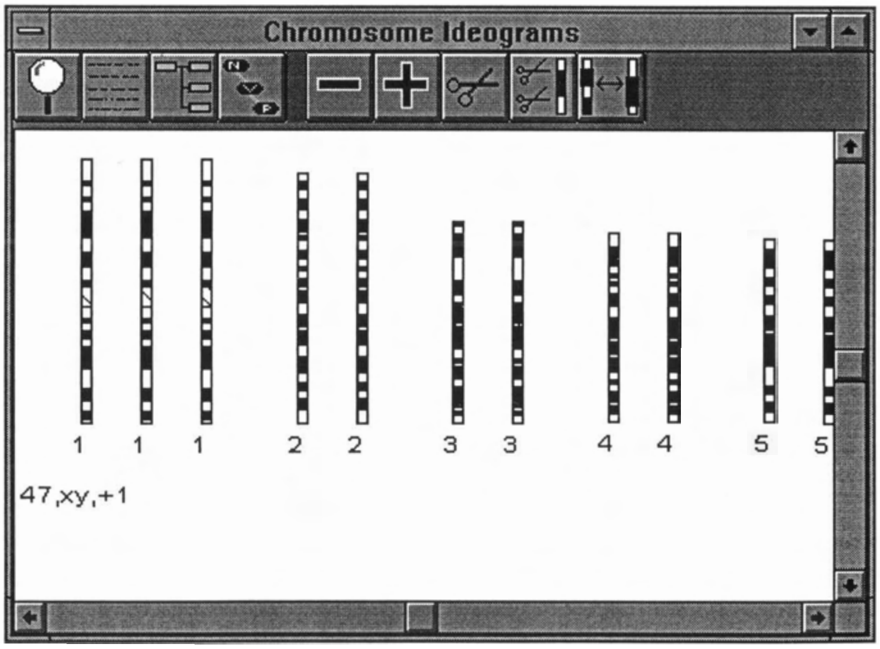
In the example of Figure 3.10 chromosome 1 is chosen for duplication and then the "+" button is pressed. The manipulator adds a "+1" to the expression, and the CBR engine is invoked to attempt to construct a new case and deep interpretation model. The matching and adaptation phases are successful and a new case is added to memory. The ideograms are then redrawn from the underlying model that was generated via adaptation (Figure 3.11).

To summarize the manipulation icons: '+' icon causes a duplication of a chromosome; '-' causes the removal; 'scissor' causes a terminal deletion of a chromosome; 'double scissors' causes the interstitial deletion of chromosome material; and 'swap' causes exchange or translocation of material.



46,xy

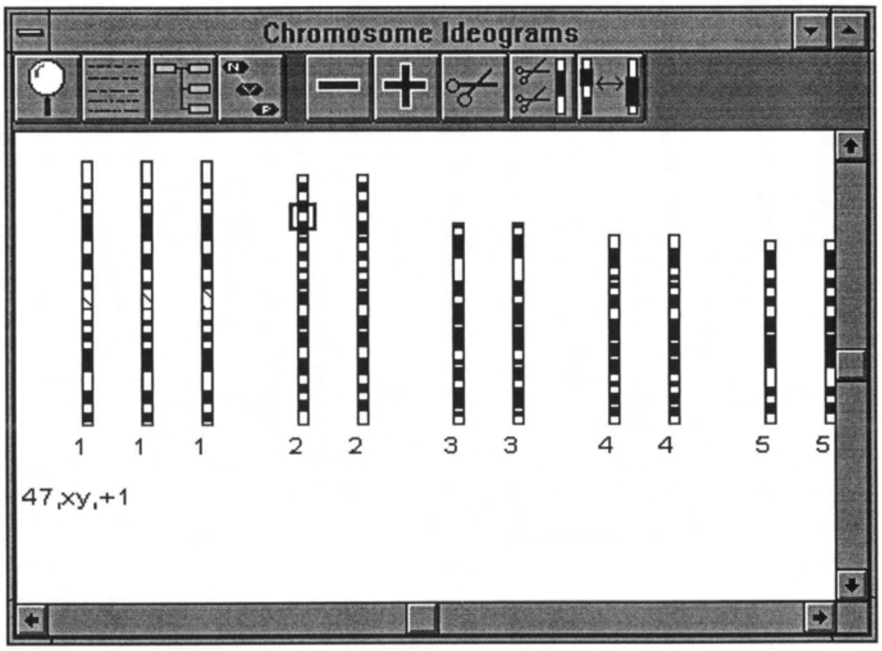
Figure 3.10 Visual Manipulation: Choosing an Entire Chromosome



47,xy,+1

Figure 3.11 Visual Manipulation: Choosing the "Add" Button Duplicates the Chromosome

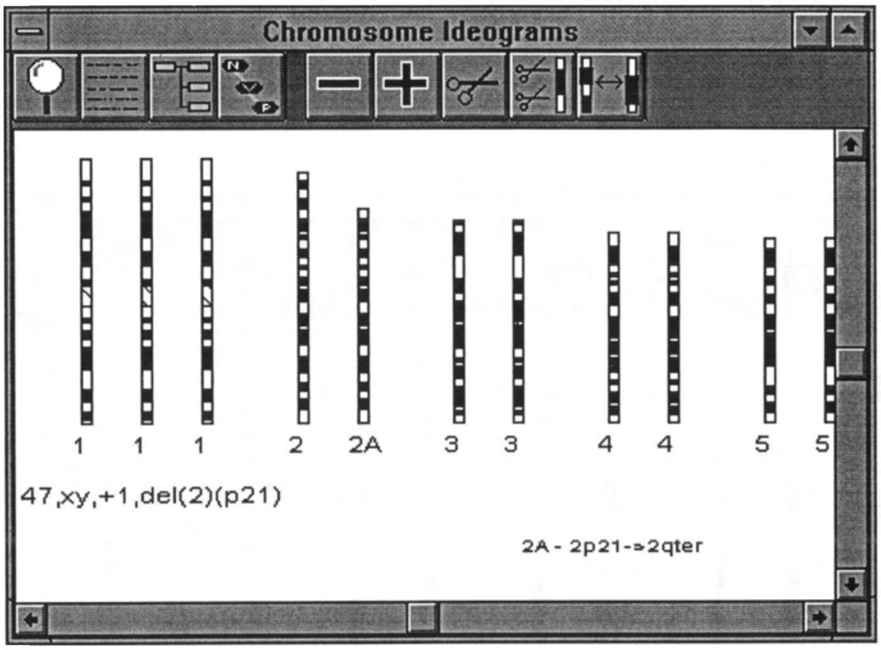
In Figure 3.12 we see another example of visual manipulation. This time a breakpoint is chosen on chromosome 2 and then the "Terminal Deletion" button is selected. Once again, the manipulator subsystem merely updates the ISCN expression with the corresponding syntactic changes. Then the CBR engine is invoked to attempt to construct a new case and interpretation model for the expression.



47,xy,+1

Figure 3.12 Visual Manipulation: Choosing a Breakpoint on Chromosome 2

An adapted case results, and the new ideogram is show in Figure 3.13. Note that the chromosome 2A is shorter, befitting a terminal deletion, and that the ISCN long form description for 2A is given on the window.

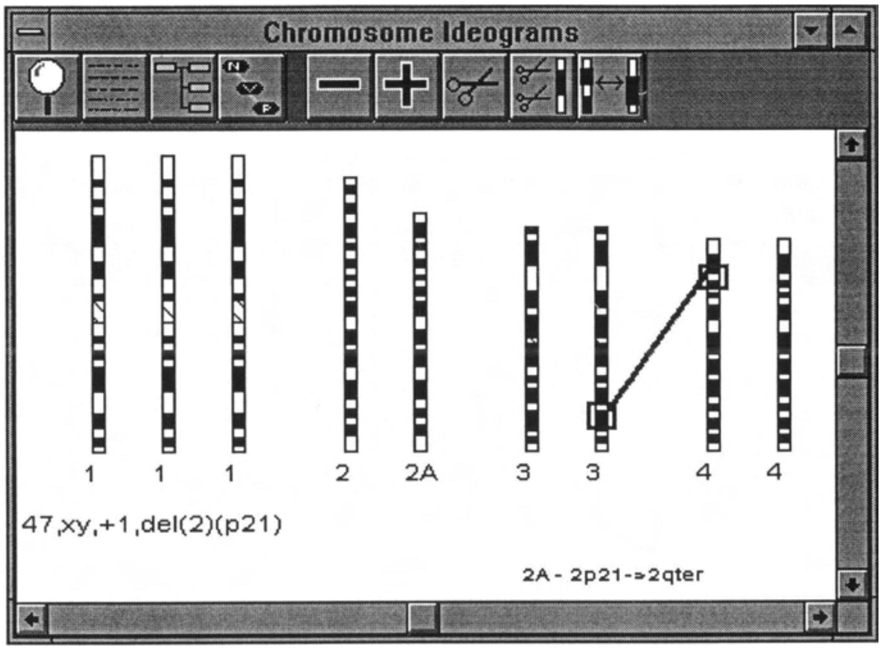


47,xy,+1,del(2)(p21)

2A - 2p21->2qter

Figure 3.13 **Visual Manipulation: Choosing the "Cut" Button creates a terminal deletion**

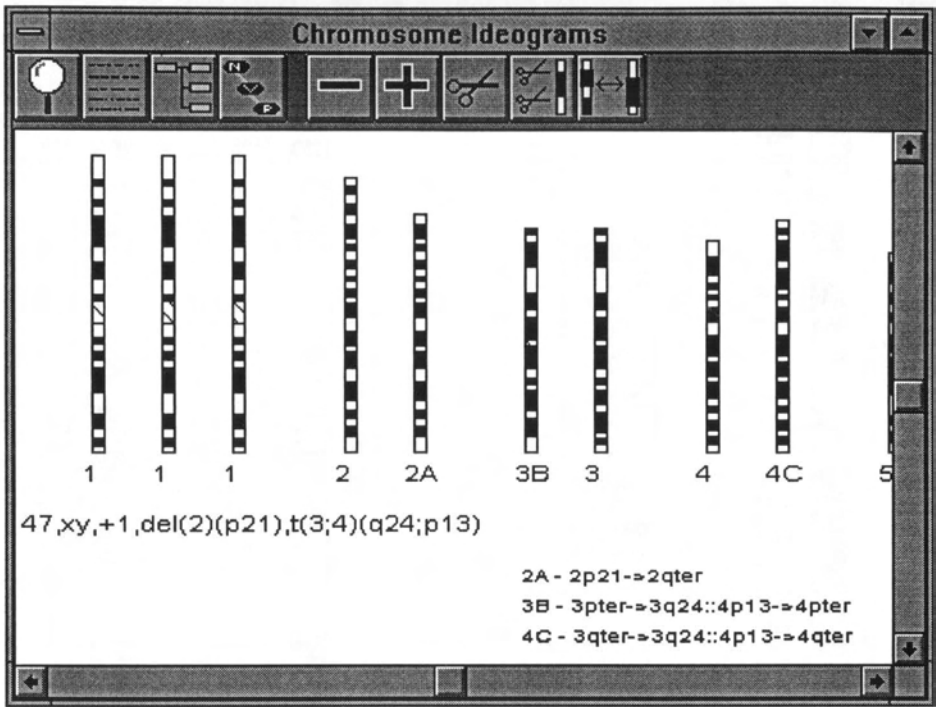
In (Figure 3.14, Figure 3.15, Figure 3.16) we see a similar pattern to the previous terminal deletion example. This time, a translocation is added via a visual manipulation, and the CBR engine once again is able to adapt a new case and interpretation, evidenced by the detailed ideogram display in Figure 3.16. Note the chromosome regions on chromosomes 3B and 4C that have been swapped, and the displayed ISCN long forms.



47,xy,+1,del(2)(p21)

2A - 2p21->2qter

Figure 3.14 Visual Manipulation: Choosing a translocation Between Chromosomes 3 and 4



47,xy,+1,del(2)(p21),t(3;4)(q24;p13)

2A - 2p21->2qter
3B - 3pter->3q24::4p13->4pter
4C - 3qter->3q24::4p13->4qter

Figure 3.15 Visual Manipulation: Choosing the "Translocation" Button creates a translocation

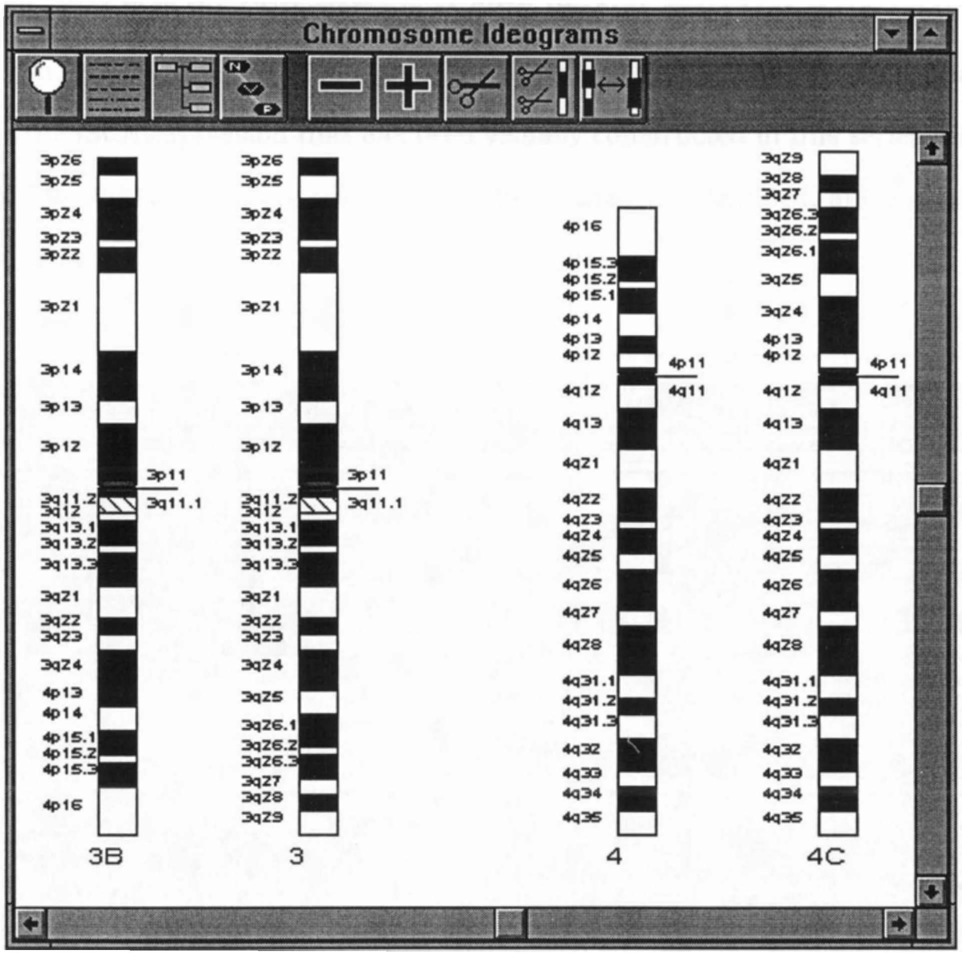
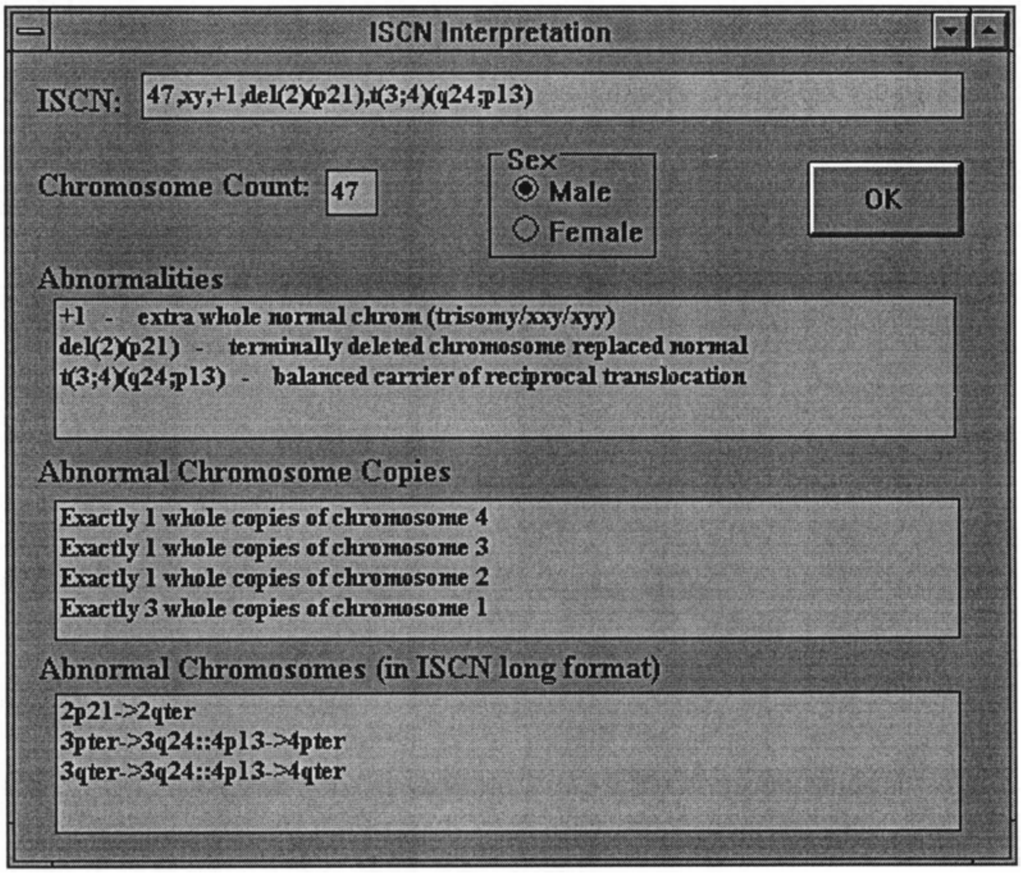


Figure 3.16 Detail of the Translocation Between Chromosomes 3 and 4 (3B and 4C)

Further proof that the CBR reasoner of ISCN Student can adapt meaningful deep interpretation (represented in MOP memory) is shown in the textual interpretation window for the ISCN expression that has been visually constructed in this series (Figure 3.17). Note that the abnormality sections have been correctly identified, and associated with meaningful descriptions.


ISCN Interpretation

ISCN: 47,xy,+1,del(2)(p21),t(3;4)(q24;p13)

Chromosome Count: 47

Sex: [✓ Male] Female

OK

Abnormalities

+1 - extra whole normal chrom (trisomy/xxy/xyy)
del(2)(p21) - terminally deleted chromosome replaced normal
t(3;4)(q24;p13) - balanced carrier of reciprocal translocation

Abnormal Chromosome Copies

Exactly 1 whole copies of chromosome 4
Exactly 1 whole copies of chromosome 3
Exactly 1 whole copies of chromosome 2
Exactly 3 whole copies of chromosome 1

Abnormal Chromosomes (in ISCN long format)

2p21->2qter
3pter->3q24::4p13->4pter
3qter->3q24::4p13->4qter

Figure 3.17 **Textual Interpretation for the Visually Manipulated Case**

An AST can be generated for expressions of arbitrary complexity, as illustrated in Figure 3.18.

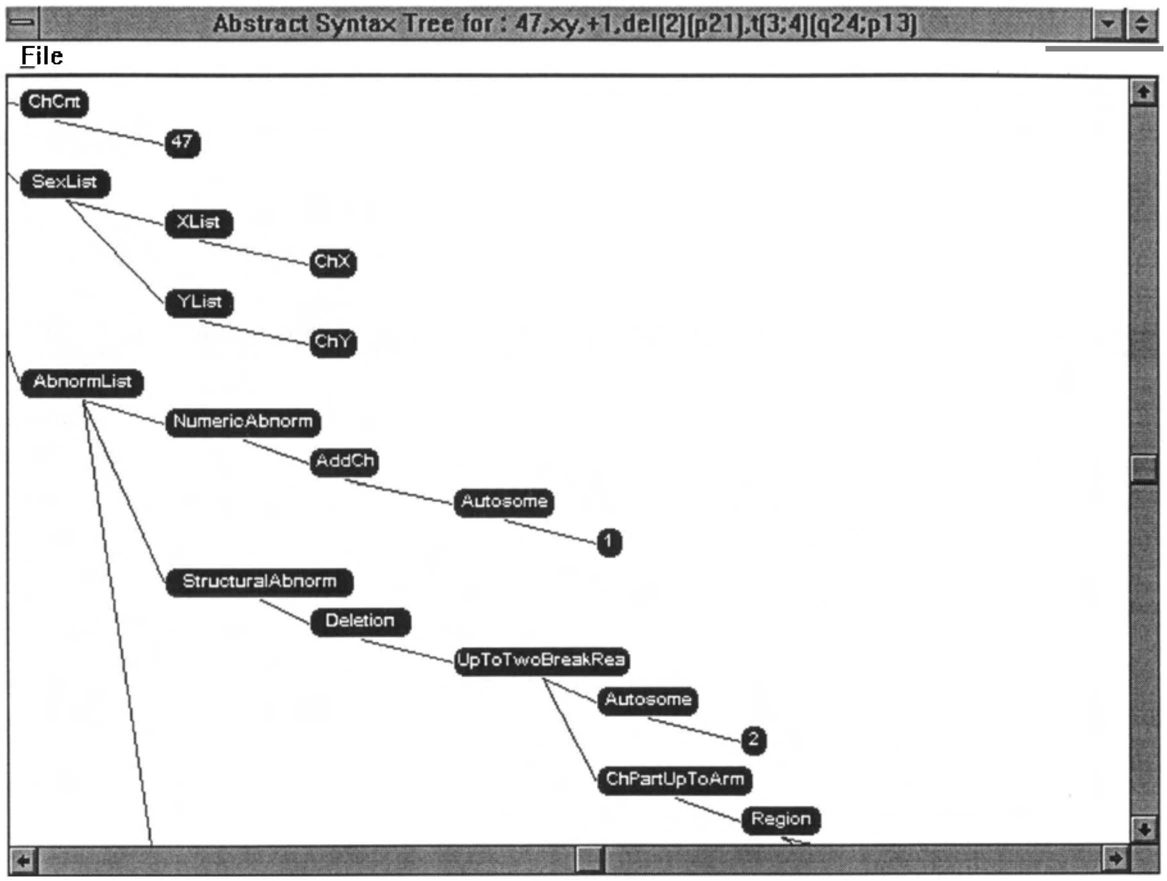


Figure 3.18 Partial Abstract Syntax Tree for 47,xy,+1,del(2)(p21),t(3;4)(q24;p13)

Finally, a view of MOP memory shows that each new ISCN case has been added with appropriate generalization inheritance relationships. Note that the following cases have been added, corresponding to the abnormalities that were introduced:

- 47,xy,+1
- 47,xy,+1,del(2)(p12)
- 47,xy,+1,del(2)(p12),t(3;4)(q24;p13)

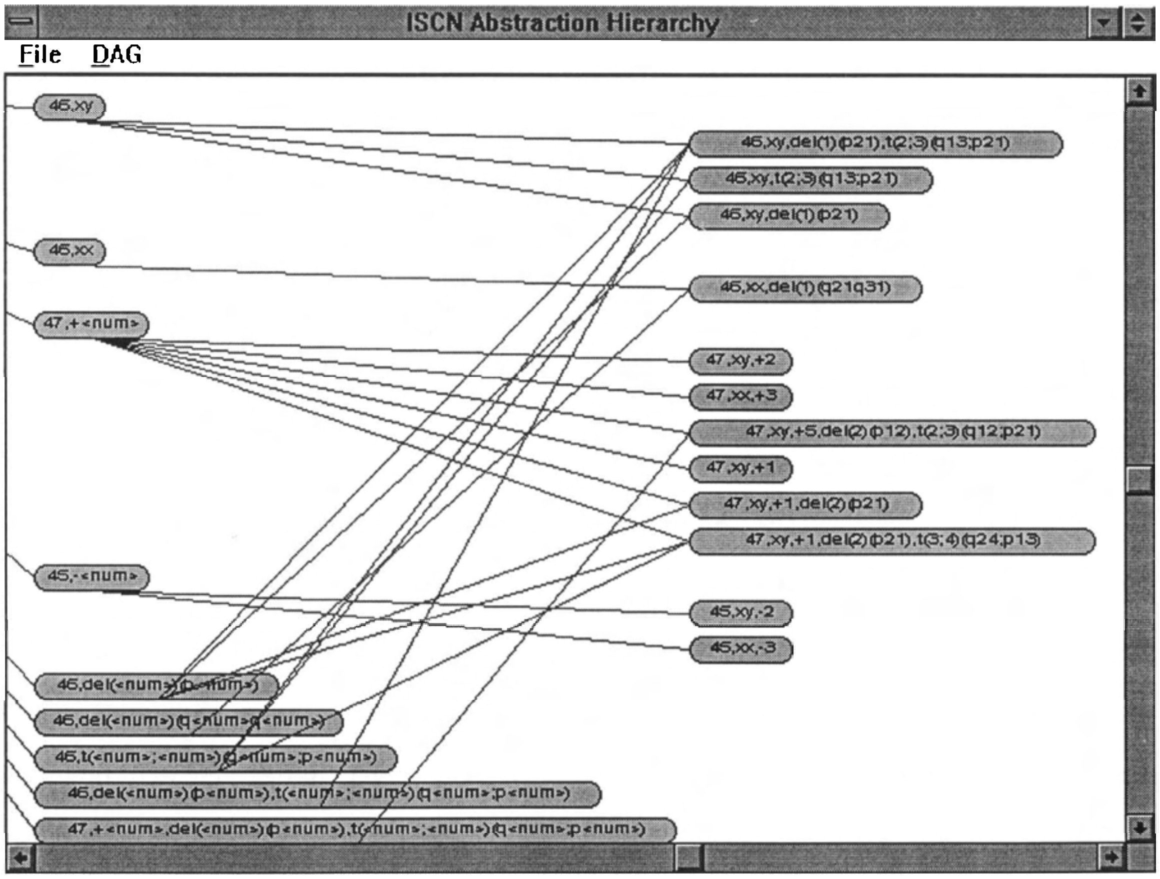

ISCN Abstraction Hierarchy
File
DAG
46,xy
46,xx
47,+<num>
45,-<num>
46,del(<num>)(p<num>)
46,del(<num>)(q<num>q<num>)
46,t(<num>;<num>)(q<num>;p<num>)
46,del(<num>)(p<num>),t(<num>;<num>)(q<num>;p<num>)
47,+<num>,del(<num>)(p<num>),t(<num>;<num>)(q<num>;p<num>)
46,xy,del(1)(p21),t(2;3)(q13;p21)
46,xy,t(2;3)(q13;p21)
46,xy,del(1)(p21)
46,xx,del(1)(q21q31)
47,xy,+2
47,xx,+3
47,xy,+5,del(2)(p12),t(2;3)(q12;p21)
47,xy,+1
47,xy,+1,del(2)(p21)
47,xy,+1,del(2)(p21),t(3;4)(q24;p13)
45,xy,-2
45,xx,-3

Figure 3.19 **Modified MOP Memory after Visual Manipulations**

4. Translating ISCN Expressions

4.1 The ISCN Grammars

The ISCN nomenclature was developed by geneticists as a notational shorthand, without consideration to grammatical consistency or parsing (Denver 1960). Fortunately, it turns out that ISCN can be expressed by an LR(1) context free grammar (CFG) (Listing 4.1), which provides the opportunity to automatically generate a parser translator using standard tools such as YACC (Johnson 1975). As ISCN Student was written in Smalltalk, a YACC-like translator generator implemented in Smalltalk was used, T-gen (Graver 1992). T-gen can construct a translator that generates both derivation and abstraction syntax trees.

There are two forms of ISCN expressions: short and long. Short form expressions indicate abnormalities, but do not explicate the actual structure of each chromosome. In contrast, long form expressions, don't state abnormalities, but do show the topology of all abnormal chromosomes.

The ISCN short form expression is at the heart of the CBR matching and adaptation, thus the grammar from which derivation and abstract syntax trees (AST) can be generated is of central importance. The resultant trees are used in several stages of the ISCN Student. One notable use is in the matching phase, where the AST of a new ISCN expression is compared to ASTs associated with existing cases. Intersection sets of AST abnormality nodes are calculated and used in ordering old cases according to closeness of match.

This (first) CFG for ISCN has been constructed to aid ISCN Student in CBR matching and adaptation. A grammar for all abnormalities was not attempted; it was limited to numerical aberrations, deletions (both terminal and interstitial) and translocations.

4.1.1 T-gen Grammar Specifications

As usual in CFG specifications, T-Gen grammars are designated by a list of productions. The left hand side (LHS) of the first production is taken to be the start symbol. T-gen supports an extension of CFGs: regular right-part grammars (LaLonde 1977), which provide improved readability and brevity in production specifications. As in YACC, T-gen notation expresses grammar productions as left hand side (LHS) non-terminals and RHS sequences of terminals, non-terminals and literals. For example,

```
Translocation :
      't' TwoChromTwoBreakRea ;
```

The non-terminal expression *Translocation* is composed of a literal 't' followed by a *TwoChromTwoBreakRea* expression

Again similar to YACC, in T-gen notation an expression in "{}" brackets at the end of production indicates a node name in an AST. For example,

```
Region :
      Arm Num            {Region};
```

This production specifies that if an AST is being generated, the parent node for an *Arm* followed by a *Num* will be named *Region* in an abstract syntax tree. The branch at the bottom of the sample AST in Figure 4.20 illustrates its realization. We see the node is labeled *Region*, with a kind of *Arm* and *Number* as terminal child nodes.

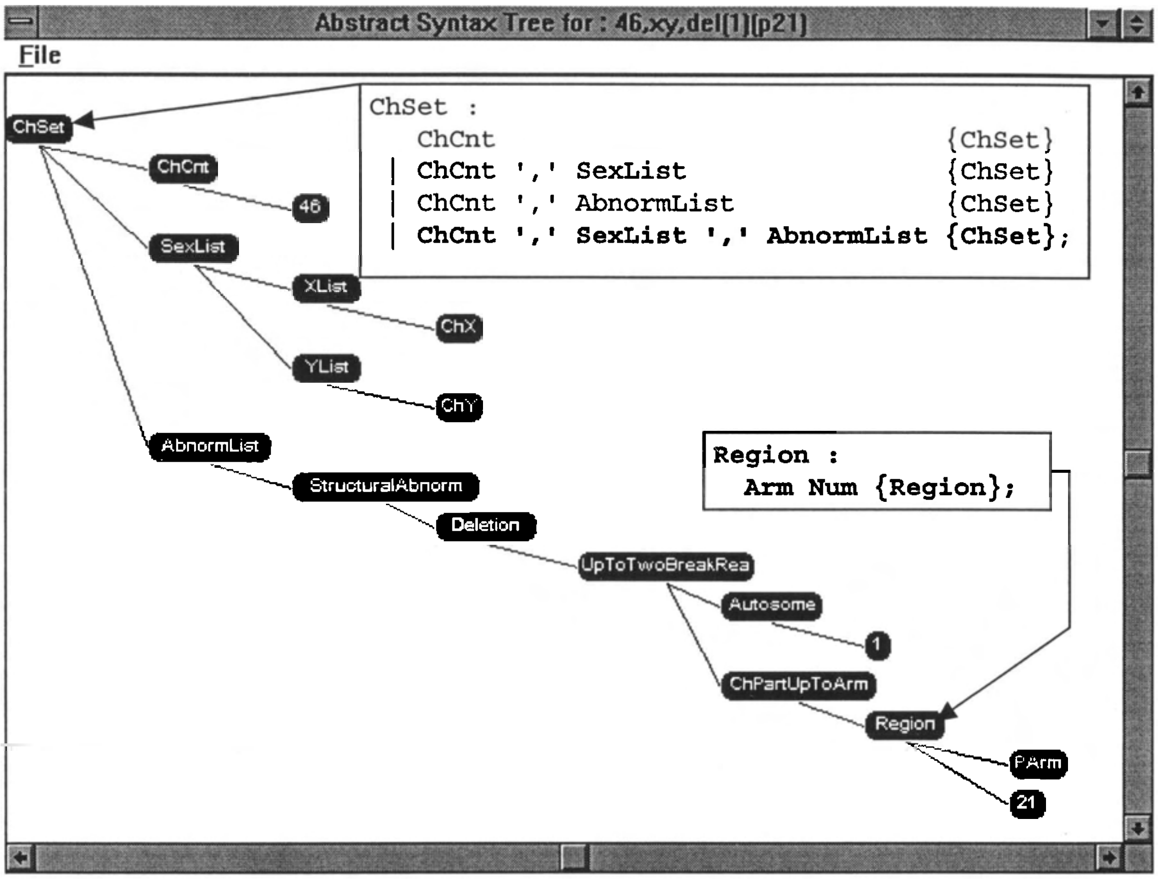
Alternate right hand side productions with the same left hand side can be consolidated with the use of the | symbol. For example,

```
ChSet :
      ChCnt                                {ChSet}
      | ChCnt ',' SexList                  {ChSet}
      | ChCnt ',' AbnormList               {ChSet}
      | ChCnt ',' SexList ',' AbnormList   {ChSet};
```

This declares that a ChSet is either a:

Alternate Production	Example
ChCnt	46
ChCnt ',' SexList	46, xy
ChCnt ',' AbnormList	46,+2
ChCnt ',' SexList ',' AbnormList	46,xy,+2

Figure 4.20 illustrates the decomposition of a *ChSet* expression into the major child branches rooted at *ChCnt*, *SexList* and *AbnormList* nodes.



```
ChSet :
    ChCnt                                {ChSet}
  | ChCnt ',' SexList                    {ChSet}
  | ChCnt ',' AbnormList                 {ChSet}
  | ChCnt ',' SexList ',' AbnormList     {ChSet};
```

```
Region :
  Arm Num {Region};
```

Figure 4.20 Abstract Syntax Tree for an ISCN Expression

4.2 ISCN Short Form Grammar

Complete listings of the short and long form grammars are given in the appendices; a subset is shown here for discussion.

```
ChSet :
        ChCnt                                  {ChSet}
        | ChCnt ',' SexList                    {ChSet}
        | ChCnt ',' AbnormList                 {ChSet}
        | ChCnt ',' SexList ',' AbnormList  {ChSet} ;

 ChCnt :
        Num                                    {ChCnt};

SexList :
        XList YList                            {SexList}
        | XList                                {SexList}
        | YList                                {SexList} ;

XList :
        ChX XList                              {liftRightChild}
        | ChX                                  {XList}    ;

YList :
        ChY YList                              {liftRightChild}
        | ChY                                  {YList}    ;

AbnormList :
        Abnorm ',' AbnormList                  {liftRightChild}
        | Abnorm                               {AbnormList} ;

Abnorm :
        NumericAbnorm                          {NumericAbnorm}
        | StructuralAbnorm                     {StructuralAbnorm} ;

" NUMERIC ABNORMS "

NumericAbnorm :
          '+'  '?' AnyChrom                    {AddUncertainCh}
        | '+'  '?' AnyChrom Sign               {AddUncertainChOfDiffLength}
        | '+' AnyChrom                         {AddCh}
        | '+' AnyChrom Sign                    {AddChOfDiffLength}
        | '-'  AnyChrom                        {MissCh}
        | '-'  '?' AnyChrom                    {MissUncertainCh}
        | '+' ChPartOfDiffLength               {AddChPartOfDiffLength}
        | ChPartOfDiffLength     ;
...
...
```

Listing 4.1 A subset of ISCN short form grammar

Consider a few examples with this grammar. The starting sentence in the grammar is:

```
ChSet :
        ChCnt                                  // alternate 1
        | ChCnt ',' SexList                    // alternate 2
        | ChCnt ',' AbnormList                 // alternate 3
        | ChCnt ',' SexList ',' AbnormList      // alternate 4
```

Alternate 1 provides a very brief karyotype description which refers only to the chromosome count:

47	a person with 47 chromosomes

Alternate 2, normal karyotypes and those defective only in sex counts are illustrated by:

46,xy	a normal male
47,xxy	a person with an extra X chromosome

Alternate 3 identifies abnormalities but excludes sex information:

46,del(1)(p21)	a person with a terminal deletion
47,+21,del(1)(p21)	a person with a terminal deletion and extra 21

Alternate 4 provides the most information on a karyotype:

48,xxy,+21	a male with an extra X and 21 chromosome

4.3 The ISCN Long Form Grammar

A grammar for the ISCN long form notation was also needed for ISCN Student to perform reasoning and interpretation of ISCN Expert output (Listing 4.2). As mentioned in the section on background concepts, the long form notation defines the detailed band-level topological structure of chromosomes; in particular, those that are abnormal.

Associated with each case in MOP memory is a set of long form expressions for each abnormal chromosome. These long form expressions are critical to the generation of chromosome MOPs that represent the structure necessary for drawing chromosome ideograms and their visual manipulation.

```
ISCNLongExpr :
        BandSection '::' ISCNLongExpr      {liftRightChild}
        | BandSection                      {ISCNLongExpr}
        | BandEnd                          {ISCNLongExpr} ;

BandSection :
        StartBand '->' EndBand             {BandSection};

StartBand :
        BandEnd                            {StartBand};

EndBand :
        BandEnd                            {EndBand};

BandEnd :
        Centromere
        | Terminal
        | Region
        | Band ;

...
...
```

Listing 4.2 A subset of ISCN long form grammar

An example with this grammar will illustrate typically long form expressions. The starting sentence in the grammar is:

```
ISCNLongExpr :
        BandSection '::' ISCNLongExpr      // alternate 1
        | BandSection                      // alternate 2
        | BandEnd                          // alternate 3
        ;
```

Alternate 1 defines the typical expression:

An abnormal chromosome that starts at the p terminal end and extends over the normal range of bands upto 1p13. This is followed by a section from 1q22 through to 1q42.

1pter->1p13::1q22->1q42

4.4 Use of Derivation and Abstract Syntax Trees in Reasoning

The definition of the ISCN grammar provides the capability to generate abstract syntax trees (ASTs) and parse (derivation) trees for each ISCN short and long-form expression associated with the CBR cases. Associated with these trees are Accessor objects, such as **ISCNExprAccessor** and **ISCNLongExprAccessor**, which encapsulate the knowledge for accessing the components of the trees (for example, the abnormality list section). These accessor objects were found to be required in order to ensure a low coupling between the components reasoning with the trees, and the tree structures and labels.

To illustrate, consider the AST in Figure 4.3 and the following code fragment used in the adaptation phase for adapting abnormalities.

```
1.    exprAccessor := ISCNExprAccessor new expr: iscnExpr.
        "// strip out the generalized ISCN abnormality expressions"
2.    genAbExprs := exprAccessor generalizedAbnormalities.
```

In reference 1, an **ISCNExprAccessor** object is instantiated on an ISCN expression string; the AST and derivation trees are generated during instantiation. In reference 2, the adaptation logic requires the subset of the expression that describes abnormalities (the **AbnormList** subtree in the figure), in a generalized form with specific chromosome numbers removed. The knowledge for the extraction and generalization of the subset expression is encapsulated within the accessor object, which delivers the results to without requiring high coupling of the reasoner (the adaptation methods) to the particulars of the tree structure.

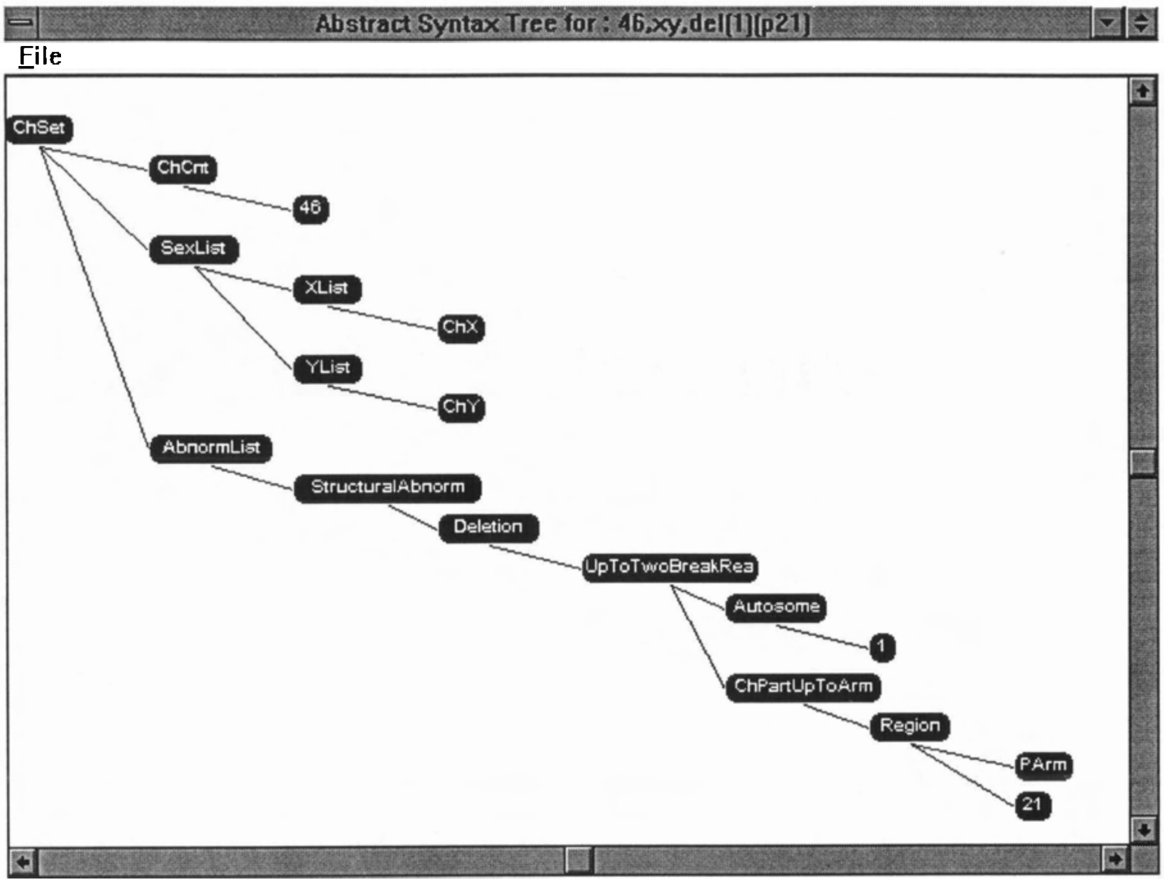


Figure 4.3 **Abstract Syntax Tree for an ISCN Expression**

5.

System Architecture

5.1 Introduction

Two dimensions of ISCN Student's architecture are discussed. The process model describes a data flow and process-oriented view which provides insight into the subsystems, system interface, and activity of the application. The object model describes the classes and their relationships for the generic CBR framework, and the class extensions required for ISCN Student.

5.2 The Process Model

New ISCN expressions may be entered manually be a user, or retrieved from a text file (Figure 5.1). ISCN Expert, an expert system written in Prolog, is invoked to process the expressions and produce textual interpretations in a new file.

These interpretations are then loaded into ISCN Student's MOP memory, a step which requires explicit knowledge of the form and content of ISCN Expert's output. A case MOP is constructed for the new interpretation, as well as MOPs for all abnormality descriptions. The MOPs are then related with associative links. Generalized abstraction (superclass) MOPs are also constructed for the case and abnormalities. Finally, all the cases and their abstractions are organized into a generalization-specialization abstraction hierarchy based on specialization by abnormality criteria. This is a CBR learning phase in which the memory is loaded with the majority of cases.

When a new expression is later entered by the user, either in textual format or via visual manipulation, ISCN Student will attempt to construct a new interpretation for it, and add it to memory. If successful, the new interpretation is presented to the user.

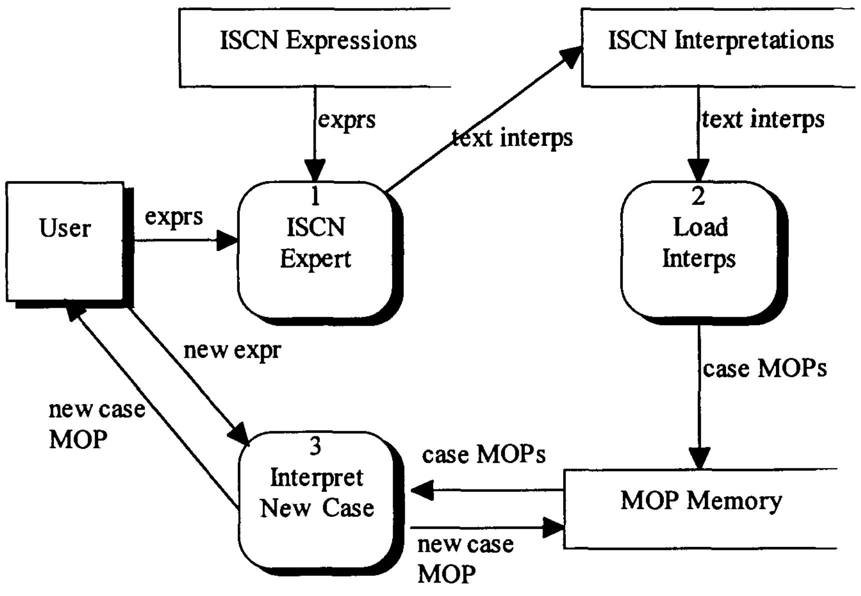


Figure 5.1 **Level 1 DFD for ISCN Student**

Taking a closer look at interpretation of new cases (Figure 5.2), ISCN Student first retrieves existing cases that most closely match the new ISCN expression, based on derivation tree or AST pattern matching; details of the matching phase will be elaborated later. The adaptation phase extracts the interpretations from the old cases, and modifies them with respect to chromosome number, abnormal chromosome structures, etc. A new case is created with the adapted interpretations, and inserted into the memory abstraction hierarchy with associative links to abnormality MOPs. If the new expression was input via the visual manipulation subsystem, the case is returned for re-display, causing the new underlying chromosome model to be used for the ideograms.

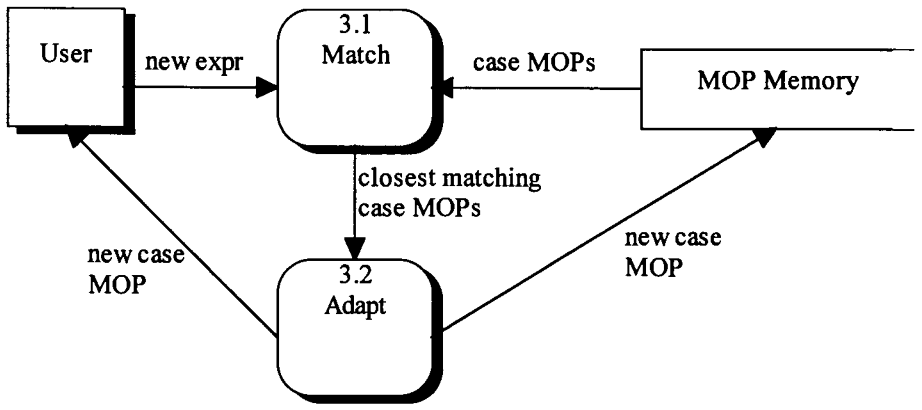


Figure 5.2 **Level 2 DFD to Interpret a New Case**

5.3 Object Model

ISCN Student was used as vehicle to explore the design of a generic object-oriented (OO) model, or *framework* (Booch 1994) for CBR systems - one that could be specialized in different domains, with a core framework of classes, attributes and methods that would prove universally useful in CBR. Furthermore, an extendible OO model for standard CBR systems provides a structure for research which builds upon existing CBR systems without needless reinvention.

The central question explored is this: what is an appropriate OO framework for a generic CBR system that supports graceful specialization in different problem domains? The answer is illustrated in Figure 5.3, which shows the object model using the extended entity relationship diagram notation of the Object Modeling Technique (Rumbaugh *et al.* 1991).

5.3.1 The Basic Object Model

The pivotal class in this model is **MOP** (Memory Organization Package), a frame-like object that is the basic unit of memory (Schank 1982). MOPs are organized into a graph structure to create a MOP-based memory which incorporates generalization-specialization hierarchies, multiple inheritance and composition relationships. A MOP is either an *abstraction* or *instance*: the former representing a class-type object, the latter a specific instance of the abstraction. An instance of the MOP class maintains the following *base* attributes:

- **name**: unique name of the MOP
- **type**: either abstraction or instance type

and the following *relationship* attributes:

- **abstractions**: a set of immediate generalizations
- **allAbstractions**: a set of all direct and indirect abstractions (used for efficient processing)
- **memory**: the associated MOP-memory object
- **slots**: a set of packaging slot relationships to other MOPs
- **specializations**: a set of immediate specializations

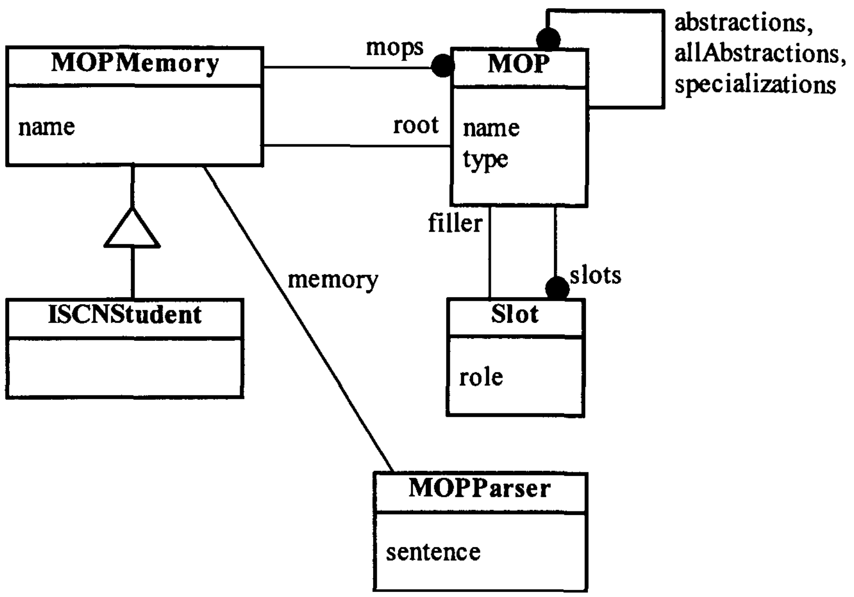


Figure 5.3 **Object model for Generic CBR Framework and ISCN Student**
(note: Solid dot represents "many" side of relationship; triangle denotes generalization-specialization)

A **MOPMemory** object is created for each problem domain, or CBR memory, reflecting the cognitive model that people possess specialized memory sets of cases; for example, criminal sentencing cases, or ISCN interpretation cases. A MOPMemory object maintains a set of all its MOPs (which can be retrieved by name), and a pointer to the **Root** MOP. MOP memories are organized into an *abstraction hierarchy* and *discrimination network* (Charniak *et al.* 1987) which is a directed acyclic graph of abstractions (generalizations) and specializations with a single ultimate ancestor MOP, **Root**, from which all other MOPs inherit. MOPMemory is a generic *abstract* class which has no (Smalltalk) instance objects, but which is specialized into *concrete* classes for each problem domain memory, such as the ISCN memory class **ISCNStudent**. Figure 5.4 illustrates the abstraction hierarchy for a portion of **ISCNStudent**.

MOPParser is a utility class to interpret structured list-form MOP definitions and store them in memory.

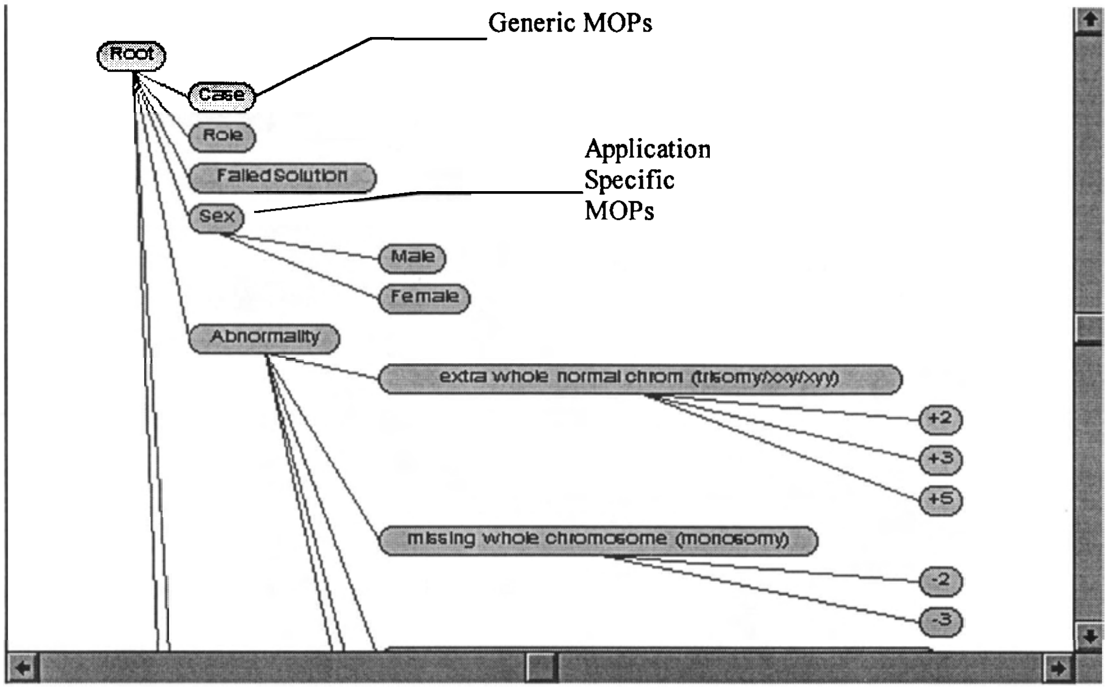

Generic MOPs
Application Specific MOPs
Root
Case
Role
FailedSolution
Sex
Male
Female
Abnormality
extra whole normal chrom (trisomy/xxy/xyy)
+2
+3
+5
missing whole chromosome (monosomy)
-2
-3

Figure 5.4 **Partial abstraction hierarchy for ISCNStudent memory**

MOPMemory is the abstract generic class for creating a CBR dynamic memory; it defines the common protocol used by all subclasses, and installs common MOPs, such as the **Root** and **Case** MOPs, into the memory object. Subclasses of **MOPMemory** define domain specific memories. For example, **ISCNStudent** is for interpretation of ISCN expressions, and installs the domain dependent MOPs, such as **ISCNInterpretation**.

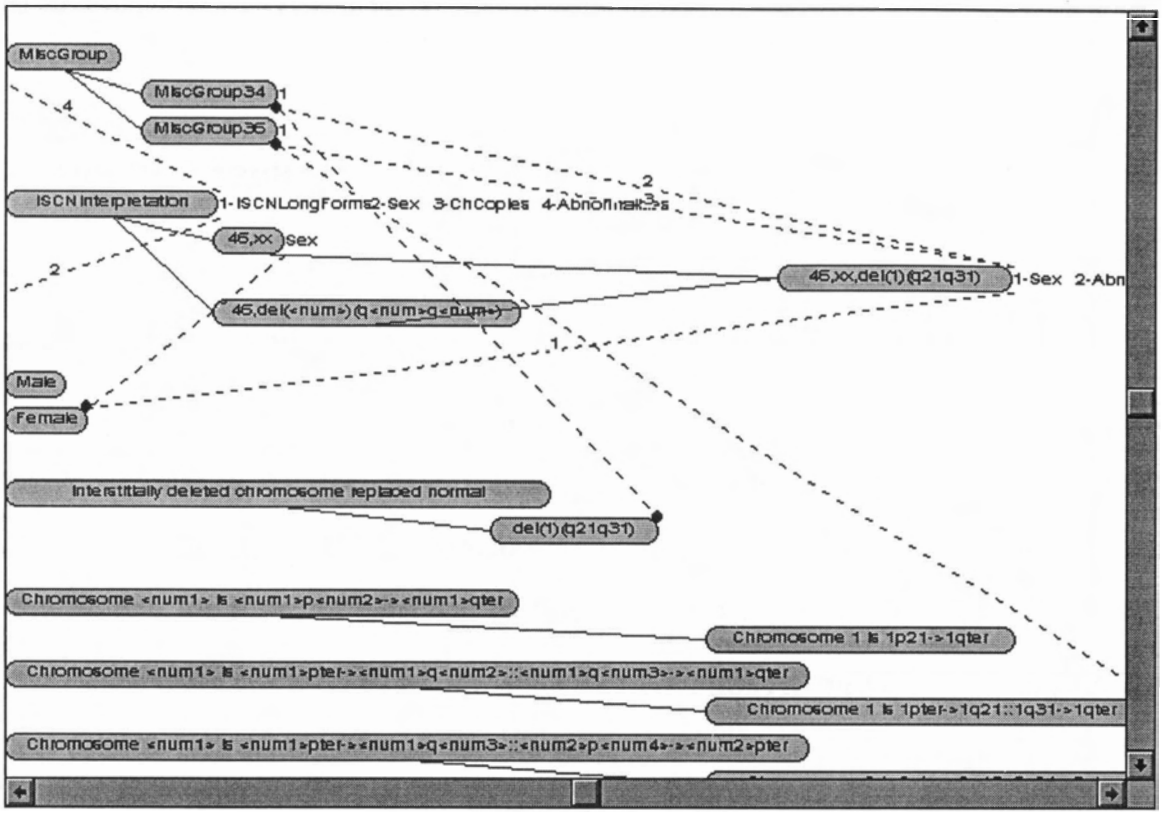

MiscGroup
MiscGroup34
MiscGroup36
ISCN Interpretation
1-ISCNLongForms 2-Sex 3-ChCopies 4-Abnormalities
46,xx
sex
46,del(<num>)(q<num>q<num>)
46,xx,del(1)(q21q31)
1-Sex 2-Abn
Male
Female
Interstitially deleted chromosome replaced normal
del(1)(q21q31)
Chromosome <num1> is <num1>p<num2>-><num1>qter
Chromosome 1 is 1p21->1qter
Chromosome <num1> is <num1>pter-><num1>q<num2>::<num1>q<num3>-><num1>qter
Chromosome 1 is 1pter->1q21::1q31->1qter
Chromosome <num1> is <num1>pter-><num1>q<num3>::<num2>p<num4>-><num2>pter
1
2
3
4

Figure 5.5. Partial associative network for an interpretation.
(note: The solid circle at the end of a dashed line indicates a directional "arrow").

A MOP has associative links to other MOPs which are represented by a set of **Slot** objects. A Slot has a *role* and *role filler* (or simply *filler*). The role is a descriptor and the filler is another MOP in the memory. For example, the **ISCNInterpretation** MOP has a slot whose role is *Sex* and whose filler is an specialization of a **Sex** MOP. Figure 5.5 illustrates the associative network for a particular case.

```
#(Outcome (Root)).

#(FightOutcome (Outcome)
        ((State PhysState) (Actor Actor) ) ).
```

Listing 5.1. Sample input structured list representation for MOPs

5.3.2 The ISCNStudent Object Model

The basic CBR framework is enhanced in ISCNStudent with classes to support interpretation and parsing of the textual descriptions.

The T-gen translator generator defines an abstract class **LR1Parser** and generates a concrete subclass **ISCNParser** class, which embodies a table representation of a state machine for the LR(1) parser for ISCN expressions. LR1Parser defines the methods for parsing (running the state machine) and creating a derivation tree.

Derivation trees are represented by a set of **DerivationTreeNode** objects, one of which is the tree root. Each node maintains a label, or symbol.

To interpret the ISCN Expert text interpretations and load them into CBR memory, an **ISCNInterpretationParser** class is defined. This class is responsible for knowing the detailed layout and semantics of the interpretations, and must create both concrete specializations and generalized abstractions of cases and abnormalities. It will be discussed in greater detail.

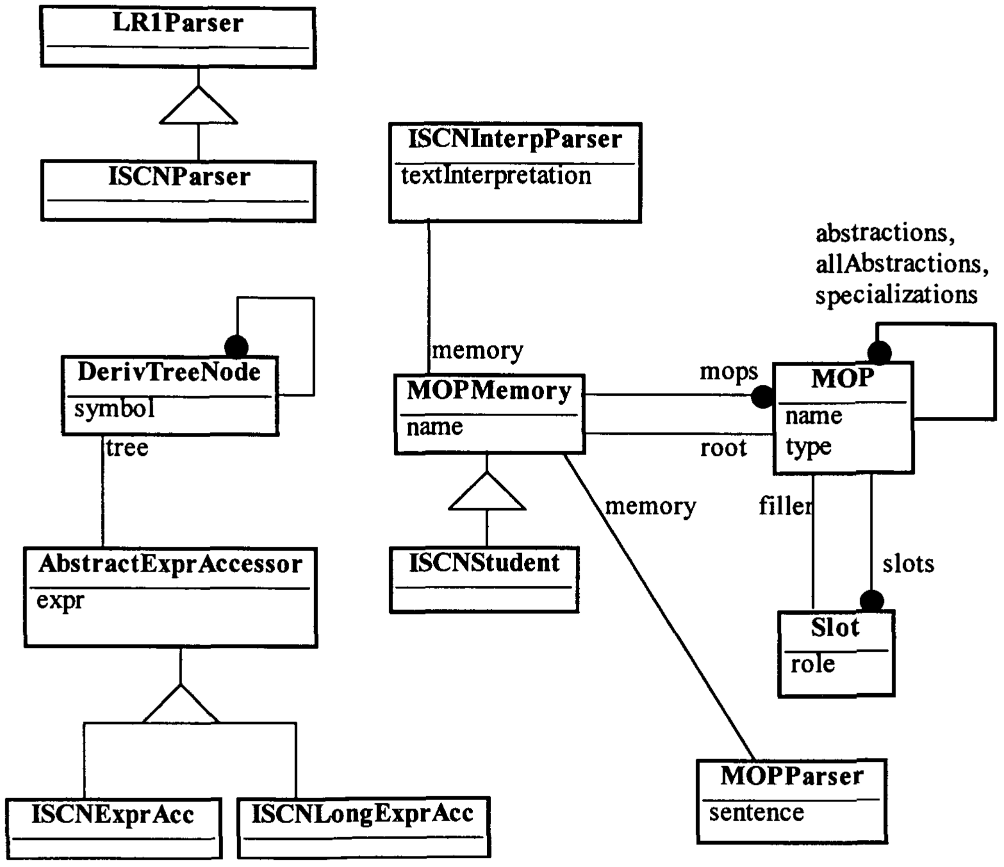


Figure 5.6 **Complete Object Model for ISCNStudent**

To support accessing the constituent components of ISCN expressions and ISCN long form expressions, which is necessary especially during the CBR adaptation phase, **ISCNExprAccessor** and **ISCNLongExprAccessor** have been defined. Objects of these classes can generate both derivation and abstract syntax trees for expressions. They possess knowledge of the structure and meaning of the expressions, and can for example retrieve the set of abnormality root nodes for those syntactic sub-units.

5.3.3 Inputting Cases

MOPs may be input using a structured list representation, which requires parsing and translation into a true Smalltalk MOP object (a detailed discussion of this process is presented in Chapter 6). For example, in Listing 5.1 the MOP **FightOutcome** is defined as a specialization of **Outcome**, with slot (*State* **PhysState**), where *State* is the role and **PhysState** the filler MOP, and slot (*Actor* **Actor**).

The translation is accomplished by a **MOPParser** object which is responsible for installing the input sentence as a MOP in a MOPMemory, with all dependent links established (abstraction, specialization and slot).

5.3.4 Inheritance in MOP Memory

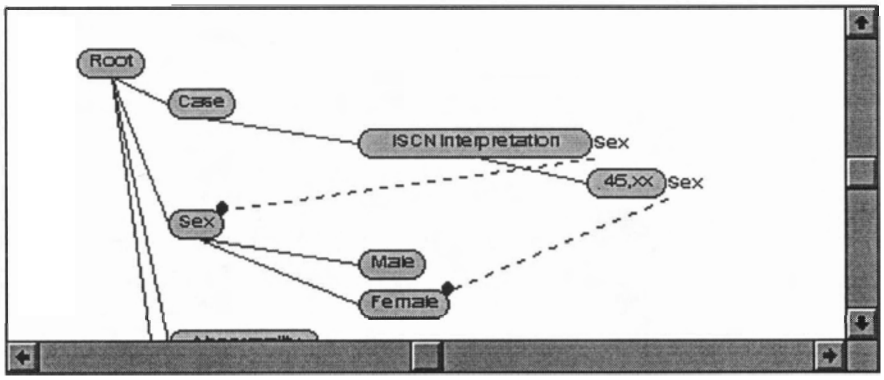
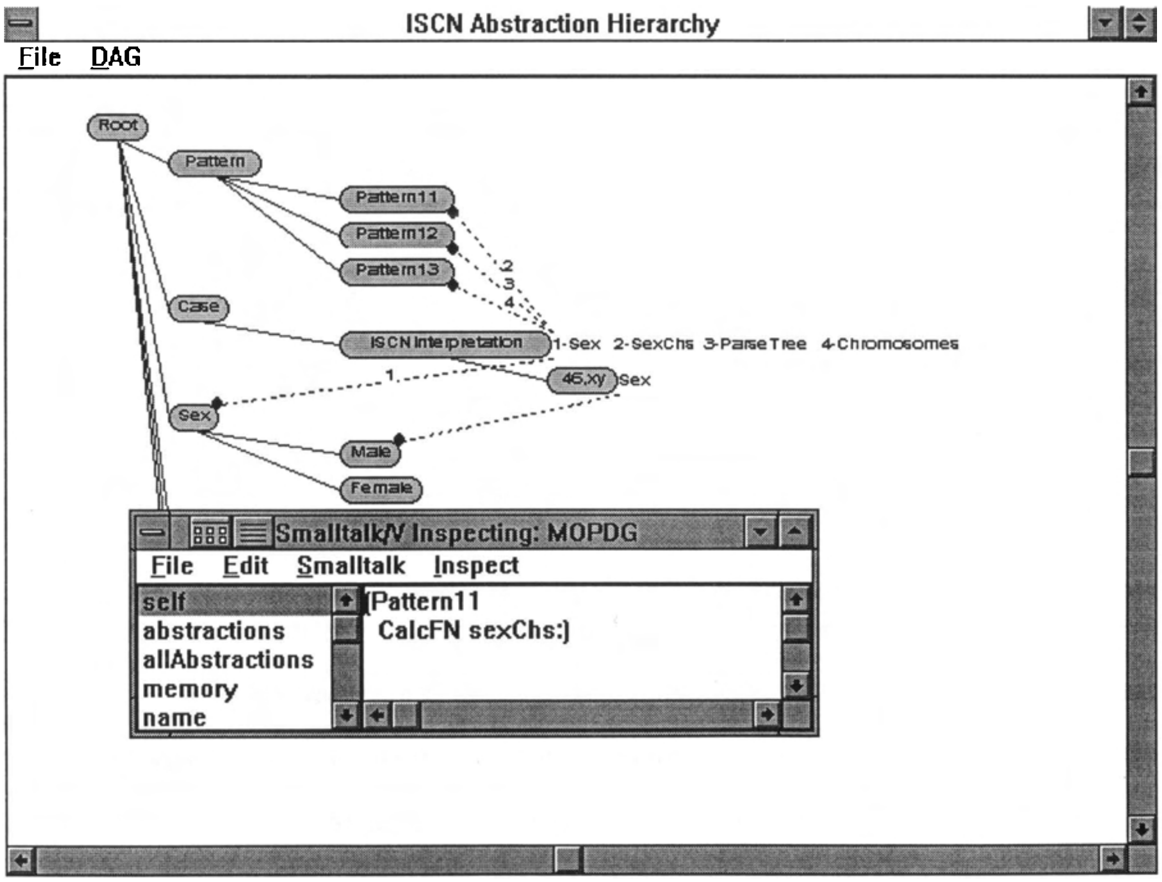


Figure 5.7. Partial associative network showing inheritance of slots

Specialized MOPs inherit the slot definitions of their abstractions in a multiple inheritance schema. For example, in Figure 5.7 the **ISCNInterpretation** MOP defines the slot *Sex*, which is associated via a slot link to the abstraction **Sex** MOP. The specialized instance MOP **46,xx** also possesses these slots and overrides the inherited fillers for more specialized ones. Thus (*Sex*-**Sex**) is replaced by (*Sex*-**Female**). Inherited slots do not need to overridden, in which case the inherited filler applies. This inheritance of values, common to frame-based languages, is in contra-distinction to pure OO languages like Smalltalk. If a MOP inherits multiple slots with the same role name, the ambiguity is resolved by using the slot of the earliest abstraction in the MOP's abstraction list.

5.3.5 Functional Attachments



```
(Pattern11
  CalcFN sexChs:)
```

Figure 5.8. Functional attachments in a MOPMemory

MOPs may have slots which contain functional attachments represented as **Pattern** MOPs; attempting to retrieve the filler for a role with such an attachment will cause the function to execute and return a filler value. These functions may have side-effects which cause I/O, pattern matching or the installation of new MOPs. They are named *Pattern* MOPs because of their use in pattern matching and role filling operations. The connections are set up as illustrated in the Figure 5.8 example: the *Sex* filler in **ISCNInterpretation** points to a Pattern MOP (**Pattern11**) which has a *CalcFN* role, whose filler is the name of an associated instance method (**#sexChs:**) in the Smalltalk language.

```
sexChs: aColl
      "MOP Calculation Function.
      Return the OC of sex chromosomes for this interpretation"
   | pattern mop acc |

   pattern := aColl at: 1.     mop := aColl at: 2.
   (mop isSuccessorOf: 'ISCNInterpretation') isSatisfied.

   acc := ISCNExprAccessor new expr: mop name.
   ^acc sexChs collect: [:aSymbol |
         (aSymbol asUpperCase includes: $X)
               ifTrue: [self at: 'Chx']
               ifFalse: [self at: 'Chy']
         ].
```

Listing 5.2. ISCNStudent Smalltalk instance method used as a functional attachment

In this framework, the location of the functional attachments was chosen to be Smalltalk instance methods (*pattern methods*) of the related **MOPMemory** Smalltalk class (e.g. **ISCNStudent**). All generic pattern functions are defined as instance methods in the **MOPMemory** class and installed in the generic memory setup. Domain specific functions are stored as methods in the related subclass of **MOPMemory**. Figure 5.8 contrasts generic functions installed in MOPMemory basic setup with those installed in the subclass ISCNStudent setup (in black).

MOPMemory defines several general purpose *pattern methods* (which are related to Pattern and Function MOPs) for testing values and generating fillers. The most important of these is **getClosestSibling:** which is a pattern matching method used to retrieve the best matching old case related to a new case, once the new case has been installed under some abstraction MOP. The generic version simply returns the first available sibling; subclasses should override this method to return a realistic best matching sibling MOP.

6.

Interpretation Parsing

6.1 Introduction

As a case-based reasoner, ISCN Student requires a set of prior cases from which to learn. These come from ISCN Expert, an existing rule-based expert system for the interpretation of ISCN expressions written in Prolog. ISCN Expert reads a set of ISCN expressions as input, and writes the interpretations to a text file, as discussed in section 5.1 (**Introduction**

Two dimensions of ISCN Student's architecture are discussed. The process model describes a data flow and process-oriented view which provides insight into the subsystems, system interface, and activity of the application. The object model describes the classes and their relationships for the generic CBR framework, and the class extensions required for ISCN Student.

The Process Model).

Text interpretations must be parsed and translated into an underlying semantic representation that captures the genetic information for each interpretation and relates it to existing MOP concepts in case memory. Thus each interpretation must be transformed into a specialization of a **Case** MOP, with MOP *roles* (akin to *frame facets*) connected to such things as **Abnormalities** and **ISCNLongForm** MOPs.

The process implies that all concepts embodied in the interpretation must be both identified and inserted into MOP memory with correct generalization-specialization and associative semantic relationships. One of the singular features of this phase is the capability to identify and generate (as MOPs) generalizations of specific concepts in a case. This *generalization generation*, and the identification of the correct placement of these concepts is rather subtle, and consumes significant effort. However, the payoff is a semantically rich case memory with the necessary generalizations and associations to provide a robust basis for the quintessential CBR capability: understanding and storing novel cases.

For example, consider the portion of memory shown in Figure 6.1. The case '47,xy,+2' as given in Listing 6.1 has just been read and interpreted. As shown in the figure, not only has the MOP named '**47,xy,+2**' been installed, but also a generalization of it named '**47,+<num>**'. The abnormality of this case is the numerical defect of a third chromosome 2 (i.e. '**+2**'). ISCN Student has installed the concept '**+2**' and has also been able to infer a generalization of it, '**extra whole normal chromosome (trisomy...)**' which has additionally been installed as a MOP. Further, the *specialized* case's **Abnormalities** attribute has been associated with the '**+2**' MOP, and the *generalized* case's **Abnormalities** attribute has been associated with the '**extra whole normal chromosome (trisomy...)**' MOP. Likewise, installation of the **ChCopies** MOP has established a generalization of it, with the specialized interpretation case linked to the specialized chromosome copies MOP, and the generalized case linked to the generalized chromosome copies MOP.

```
47,xy,+2
```

```
1 2 3 4 5 6 7 8 9 10111213141516171819202122x y
| | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | |
  |
Sex model is male
Exactly 3 whole copies of chromosome 2
XNX(2) = extra whole normal chrom (trisomy/xxy/xyy)



47,xy,+5,del(2)(p12),t(2;3)(q12;p21)

1 2 3 4 5 6 7 8 9 10111213141516171819202122x y
|   | | | | | | | | | | | | | | | | | | | | | |
|     | | | | | | | | | | | | | | | | | | |
        |
Cell observation is reciprocal translocation
Cell observation is terminal deletion
Chromosome 2 is 2pter->2q12::3p21->3pter
Chromosome 3 is 2qter->2q12::3p21->3qter
Chromosome 2 is 2p12->2qter
Sex model is male
Exactly 1 whole copies of chromosome 3
Exactly 0 whole copies of chromosome 2
Exactly 3 whole copies of chromosome 5
RTB(2,3) = balanced carrier of reciprocal translocation
TDR(2) = terminally deleted chromosome replaced normal
XNX(5) = extra whole normal chrom (trisomy/xxy/xyy)
```

Listing 6.1 Sample input text file of two interpretations from ISCN Expert

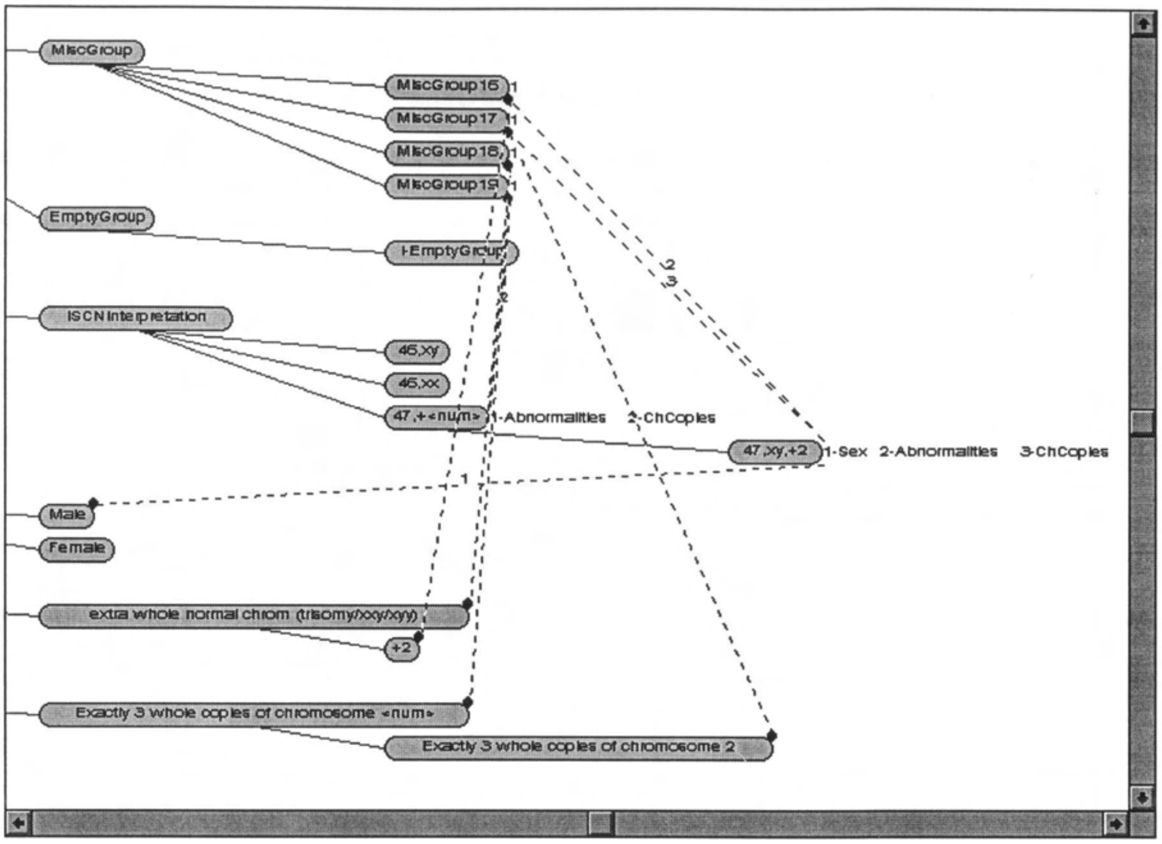

MiscGroup
MiscGroup16
MiscGroup17
MiscGroup18
MiscGroup19
EmptyGroup
I-EmptyGroup
ISCNInterpretation
45,xy
45,xx
47,+<num>
1-Abnormalities 2-ChCopies
47,xy,+2
1-Sex 2-Abnormalities 3-ChCopies
Male
Female
extra whole normal chrom (trisomy/xxy/xyy)
+2
Exactly 3 whole copies of chromosome <num>
Exactly 3 whole copies of chromosome 2

Figure 6.1 **MOP memory illustrating specialized and generalized cases.**

6.2 Solution Analysis

Objects of class **ISCNInterpretationParser** are responsible for performing the translation. The top level method is **#loadInterpretationsFrom:,** which reads and transforms the interpretations (Listing 6.2). The algorithm involves reading through the text file, extracting each discrete interpretation, and transforming it. Inside the file iteration logic are the essential steps:

```
[mops add:
      (self calcMOPModelFor:
            (self nextInterpretation: inStream))].
```

In this expression, method **#nextInterpretation**: is invoked to extract and return a collection of strings comprising an interpretation. Method **#calcMOPModelFor**: then transforms it into a MOP in case memory, and returns the results, which are added to a set of new cases. Finally, in the line:

```
(self mopMem at: 'ISCNInterpretation') reorganizeAllSpecializations.
```

the generalization-specialization relationships of all MOPs below **ISCNInterpretation** (the abstract class of all case interpretations) are restructured in memory to reflect the influence of the new cases.

```
loadInterpretationsFrom: fileName
    "Read a set of ISCNXpert interpretations from a file, and
    transform each into a MOP structure in MOP memory. Return
    all the mops created."
        | inStream mops |

    mops := OrderedCollection new.
    inStream := File pathName: fileName.
    [inStream atEnd] whileFalse:
        [mops add:
            (self calcMOPModelFor:
                (self nextInterpretation: inStream))].
    inStream close.

        "// establish generalization patterns based on abnormalities"
    (self mopMem at: 'ISCNInterpretation') reorganizeAllSpecializations.
    ^mops.
```

Listing 6.2 Top level method for reading and creating cases

The interesting work for a single interpretation is controlled by **#calcMOPModelFor**: (Listing 6.3). This method drives the invocation of the sub-tasks which both parse out each ISCN interpretation feature, and which generate generalizations and associations.

In references 1 through 6 the transformation for each semantic unit in the text interpretation is performed.

For example, in reference 2 **#parseOutChCnt** processes chromosome count information.

Reference 7 composes the new case MOP.

Reference 8 installs it in case in memory.

Reference 9 causes a demon to execute which generates a generalization of the new case, installing it too in memory.

```
calcMOPModelFor: strings
        "Parse strings, which contains an ISCNXpert interpretation,
        and store it as a MOP structure in MOP memory. Answer the
        MOP created. Also, generate generalized interpretations and
        store them in memory."
    | mopStruct slots mop x |

    self textInterpretation: strings.

    mopStruct := OrderedCollection new.
    slots := OrderedCollection new.
    slots
1.         add: (Array with: #Expression with: self parseOutISCNExpr);
2.         add: (Array with: #ChCnt with: self parseOutChCnt);
3.         add: (Array with: #Sex with: self parseOutSexModel).
4.      x := self parseOutAbnormalityDescriptions.
        x isEmpty ifFalse:
             [slots add:(Array with: #Abnormalities with: x)].
5.      x := self parseOutChCopies.
        x isEmpty ifFalse:
             [slots add:(Array with: #ChCopies with: x )].
6.      x := self parseOutLongFormAbnormalities.
        x isEmpty ifFalse:
             [slots add:(Array with: #ISCNLongForms with: x)].

7.      mopStruct
          add: self parseOutISCNExpr;
          add: #(ISCNInterpretation) ;
        add: #abstraction;
        add: slots.

        "// load the structure into memory, and then generate
            a generalized abstraction of it (a learning step)"
8.    mop := self mopMem storeCollAsMOP: mopStruct asArray.
9.    mop fillerFor: 'GeneralizedInterpretation'.
      ^mop
```

Listing 6.3 Method to transform a single text interpretation into a case MOP

Transforming a semantic unit into MOPs, creating associations and concept generalization is performed by the **#parseOut<Unit>** methods, such as **#parseOutChCopies** (Listing 6.4). This example is the simplest of the set, but serves for illustration.

In reference 1, the text lines specific to chromosome counts are extracted.

Reference 2 transforms the chromosome count expression into a generalization concept, in this case by replacing the specific chromosome with a variable.

At reference 3 the extracted specific chromosome count concept and the generated generalized concept are installed in memory, in a specialization hierarchy rooted at the generic MOP concept **ChCopy**.

Reference 4 returns the collection of specialized chromosome copy MOPs, for establishing associative links to the new case MOP.

```
parseOutChCopies
        "Return the statements containing the copies info, after storing
        them and generalized abstractions in memory."
    | copies x genCopies |

        "// get the raw text lines"
1.    copies := self textInterpretation select: [:line |
            line hasSubCollection: 'copies of chromosome'].

        "// create generalized abstractions.
            e.g. exactly 3 copies of <num> "
2.    genCopies :=copies collect: [:aString |
            x := aString asArrayOfSubstrings.
            x at: x size put: '<num>'.
            x asStringOfElements].
        "// store the gen and spec lines as MOPs in memory"
3.    copies with: genCopies do: [:copyInfo :genCopyInfo |
            self mopMem storeStringAsMOP: genCopyInfo under: #(ChCopy).
            self mopMem storeStringAsMOP: copyInfo
                                   under: (Array with: genCopyInfo).
            ].

4.    ^copies collect: [:line | Array with: line].
```

Listing 6.4 Method to parse out chromosome counts and create generalization

The final stage in loading cases from ISCN Expert is creating a generalization of the new case itself (Listing 6.5). In simplified terms, the generalization involves the replacement of specific chromosomes by variable patterns, and associative links to abstract generalizations of abnormalities rather than concrete genetic defect MOPs.

In reference 1, a new generalized version of the ISCN expression is created. An expression accessor object is generated for the ISCN expression; it can created syntax trees and knows the structure of the expressions. The sex symbols are removed, the chromosome count is instantiated rather than treated as a variable, and all chromosome references are treated as variables.

At reference 2 the slots of the new generalized interpretation MOP are established.

In references 3 and 4 the abnormalities of the specific case are extracted and their generalizations are collected. These generalized abnormality MOPs are then attached via associations to the new generalized interpretation MOP.

References 5 and 6 perform similar operations with chromosome copy and ISCN long form information; establishing relationships to generalization concepts of the specific case details.

At reference 7 the new MOP is declared a specialization of a MOP **ISCNInterpretation**, and at reference 8, the new MOP is installed in case memory.

```
generalizedInterpretation: aColl
      "MOP Calculation Function.
      Return the generalized interpretation of an ISCN case. Used in
      'learning' new generalizations which get remembered.
      e.g. 46,xy,+2 returns 46,+<num> with associated generalized
      abnormalities, etc."
   | acc pattern mop mopForm slots genMOP newExpr tree
      sexList chCnt abs copies longForms|

   pattern := aColl at: 1.    mop := aColl at: 2.

   mopForm := OrderedCollection new.
   slots := OrderedCollection new.

      "// gen the new generalized iscn expression. eliminating the
         sex if the sex is normal. We'll get
         something like:  45,-<num>  "
1.   newExpr := (acc := ISCNExprAccessor new
                     expr:  (mop fillerFor: 'Expression'))
         removeSex;
         instantiateDerivTreeChCnt;
         generateDerivTreeSentence.

      "// don't bother with the special case of having
         only a chCnt left (e.g. '46')  "
   newExpr size = 2 ifTrue: [^nil].

   tree := acc derivTree.

      "// if a mop by this name already exists, use it instead,
         and update its relationships"
   (genMOP := self at: newExpr) notNil ifTrue: [
      genMOP makeSuitableSiblingAbstractionsIntoKids.
      ^nil].

      "// load up the slots"
2.   slots
         add: (Array with: #Expression with: newExpr);
         add: (Array with: #ChCnt with:(mop fillerFor: 'ChCnt'));
         add: (Array with: #DerivationTree with: tree).

         "// find the generalization of the specific abnormalities"
3.   abs := (mop fillerFor: 'Abnormalities') groupMembers.
4.   abs isEmpty ifFalse: [
         slots add: (Array
                           with: #Abnormalities
                           with: (abs collect: [:abnorm |
                                 abnorm abstractions first]))].

         "// find the generalization of the specific ch copies info"
5.   copies := (mop fillerFor: 'ChCopies') groupMembers.
     copies isEmpty ifFalse: [
            slots add: (Array with: #ChCopies
                              with: (copies collect: [:copyInfo|
                                    copyInfo abstractions first]))].

            "// find the generalization of the specific ISCNLongForms"
6.    longForms := (mop fillerFor: 'ISCNLongForms') groupMembers.
      longForms isEmpty ifFalse: [
            slots add: (Array with: #ISCNLongForms
                            with: (longForms collect: [:longForm|
                                    longForm abstractions first]))].

7.    mopForm
        add: newExpr;
        add: #(ISCNInterpretation);
        add: #abstraction;
        add: slots.

      genMOP := MOPParserDG defMOP: mopForm in: self.
 8.   genMOP installConcept.
      ^nil.
```

Listing 6.5 Demon for creating generalization MOPs of new case MOPs

6.3 MOP Installment and Automatic Memory Reorganization

The last step of installing the new MOP in memory, is generally invoked via the message:

```
      aMOP installConcept
```

Method #installConcept (Listing 6.6) causes a reorganization of affected MOPs with respect to generalization-specialization relationships. For example, the introduction of a new generalized interpretation case MOP may require that conceptual specializations of it that are not yet related, be linked up. This process is generic to all case memories, and is sufficiently subtle and powerful that it warrants a more full investigation.

The method installs both instance and abstraction MOPs. Consider the case for abstractions. First, at reference 1, #installAbstraction is invoked, which causes sibling instance MOPs to become specializations of the new abstraction, if they are logically conceptual specializations. Last, at reference 2, sibling abstractions that should also be specializations of the new abstraction are also reorganized as children.

Taking a closer look at **#installAbstraction**, it first determines if there is another *twin* abstraction (possibly under a different name) that is semantically identical. If so, the new MOP is removed and ignored.

If the MOP is really a new concept, reference 4 causes suitable sibling instances are reorganized as specializations.

```
installConcept
        "Place myself at an appropriate place in the memory hierarchy,
            moving siblings to be my kids if appropriate."
    | abs |

    self type notNil isSatisfied.

    self isInstance
        ifTrue: [self installInstance]
        ifFalse: [
1.                abs := self installAbstraction.
2.                abs makeSuitableSiblingAbstractionsIntoKids].



installAbstraction
    "Place an abstraction at its appropriate place in the hierarchy"
        | twin |

      twin := self getTwin.
        "// if there's a twin, don't bother with me"
3.    twin notNil ifTrue: [self remove.  ^twin].
4.    self makeSuitableSiblingInstancesIntoKids.
      ^self.
```

Listing 6.6 Top level methods to install new MOPs and reorganize memory

Making suitable siblings that are instances into specializations of the new abstraction is handled in Listing 6.7. At references 1 and 2, siblings are collected.

In reference 3, abstractions are filtered out, leaving only instance MOPs.

Reference 4 contains the most interesting test: is the new MOP a suitable abstraction of an existing instance?

If the new abstraction should be a parent of an existing instance, then at references 5 and 6 the instance in unlinked from its old parent abstraction and defined to be specialization of the new one.

```
makeSuitableSiblingInstancesIntoKids
    "re-index all sibling instances as specializations of me,
      if they can be"

1.      self abstractions do: [:parent |
2.          parent specializations do: [:sibling |
3.              (sibling isInstance and:
4.              [self isSuitableAbstractionOf: sibling]) ifTrue: [
5.                      sibling unlinkFromAbstraction: parent.
6.                      sibling linkToAbstraction: self]]].
        ^self.
```

Listing 6.7 **Method to reorganize memory when installing new abstractions.**

The test to determine if a mop should be an abstraction of another is handled by **#isSuitableAbstractionOf**: (Listing 6.8). We first distinguish between MOPs which are collections or groups, and those that are not. A **Group** MOP *A* can be an abstraction of another group MOP *B* if *A* recursively contains members that are suitable abstractions of the members of *B*. This test is handled within the block starting at reference 1.

The more common non-group case begins at reference 2. At reference 3 the slots that can be used in this generalization classification are extracted for testing. It was discovered during development that MOP concepts need attributes (*slots*) that may not participate in generalization tests, thus only a subset of mops (classification slots) are eligible candidates for this testing. Each classification slot must satisfy (a predicate calculus *for-all* condition) the test beginning at reference 4.

Reference 4 tests that the MOP under consideration (*mop*) as a specialization has the same role (or attribute) as the potential abstraction (*self*).

At reference 5 we test that the attribute values (or *fillers*) for identical attributes in the two MOPs pattern match. The pattern match testing method (**#hasMatching:to:of:**) includes obvious cases such as equality, and more interestingly, a case where the 2 values (which may themselves be MOPs) are recursively tested using **#isSuitableAbstractionOf:** to determine if one value is a potential abstraction of the other.

```
isSuitableAbstractionOf: mop
    "Answer true if all my slots are 'satisfied' by the
      corresponding slots in mop. If satisfied, I am
      a suitable parent of mop. This is the major
      pattern matching method that aids in the placement
      of new input MOPs.

    *** If 2 group MOPs are being compared, I DON'T require that the
    contained MOPs be in corresponding slots (e.g. 1&1, 2&2), but
    rather that each contained MOP in self is an abstraction of any
    MOP in group 'mop' "

    (self == mop) ifTrue: [^false].
    (self isAbstractionOf: mop) ifTrue: [^true].
    (self isInstance or: [self hasSlots not]) ifTrue: [^false].

    (self isGroup and: [mop isGroup])
1.        ifTrue: [
            ^self groupMembers alwaysSatisfies: [:myMember |
                 mop groupMembers has: [:hisMember |
                       myMember isSuitableAbstractionOf: hisMember]]]
2.        ifFalse: [
3.          ^self classificationSlots alwaysSatisfies: [:mySlot |
                 "// mop must have the roles I have, and the fillers
                       of mop and I must pattern match"
4.               (mop hasRole: mySlot role)
5.               and: [self hasMatching: mySlot filler
                                      to: (mop fillerFor: mySlot role)
```

```
                                          of: mop]]]
```

Listing 6.8 **Predicate method to evaluate potential generalization-specialization between 2 MOPs**

7. Matching

In CBR systems, matching is the process of finding the most similar existing cases to a new case. Ideally, a perfect match is found and thus solutions need not be adapted (*null adaptation*). More likely is that adaptation will be required. The matching strategy of ISCN Student is to find a set of closest matching cases that collectively cover all the abnormalities present in the new case, and then a synthesis of the old interpretations is used to construct a new interpretation.

In order for retrieval to work successfully, the *indexing problem* (Kolodner 1993) must be addressed; that is, the problem of efficiently retrieving applicable cases at the right time. This problem includes the *similarity-assessment problem* of how to recognize when an existing case is applicable to a new situation, the *indexing-vocabulary problem* of choosing appropriate generalized abstractions to aid in comparing cases, the *ranking problem* of ordering similar cases according to a goodness of match measure, and the *retrieval problem* of efficient search in large search space (case library).

In order to achieve matching, ISCN Student must provide a solution for retrieval and ranking. The generic OO framework provides a default pattern method **MOPMemory>>getClosestSibling** which simply returns the closest sibling in the graph of memory, based on discrimination by abstraction. Normally this method is overridden in subclasses, or an alternate is used. ISCN Student uses an alternate method, **ISCNStudent>>matchSetTo:**, which returns a set of existing cases that best match to the new case, ranked in decreasing order of goodness of match.

```
interpretNewCaseFor: iscnExpr
        "Answer a new (or existing) case based on old ones.
        Attempt to interpret a new expr string based on the existing
        cases. This is the main 'new case' learning phase. If the
        expr already has a case, return it."
    | closestOld new matches case |

            "// already got a case? "
    (case := self at: iscnExpr) notNil ifTrue: [^case].

            "// search for matches to generalizations of old cases
                based on parse tree intersections of abnormalities."
    matches := self matchSetTo: iscnExpr.
    (matches includes: nil) ifTrue: [^nil].

            "// construct a new case from the old matching ones."
    ^self adapt: matches forExpr: iscnExpr.
```

Listing 7.1 Top Level Method for Case Interpretation. Performs Matching and Adaptation

ISCNStudent>>interpretNewCaseFor: (Listing 7.1) is the primary method used to construct new case solutions. The statement:

```
matches := self matchSetTo: iscnExpr.
```

invokes the **ISCNStudent>>matchSetTo:** method which returns the matching set. The adapted case is constructed in the final statement:

```
^self adapt: matches forExpr: iscnExpr.
```

which takes the set of closest matches and new expression, returning a complete specialization of an **ISCNInterpretation** MOP (the adaptation phase is described in the next chapter).

ISCNStudent>>matchSetTo: (Listing 7.2) is responsible for identifying and returning the closest existing case MOPs that provide all of the interpretation material needed to interpret the new case. This new case is represented as the argument to the method - as a string containing the expression. The approach is to find an existing case that includes the closest matching interpretation for each constituent abnormality in the new expression. As each constituent abnormality is resolved, its grammatical representation is removed from the expression, and a recursive invocation solves for the remaining sentence.

It might be invoked as follows:

```
matches := self matchSetTo: '46,xx,+2,del(3)(p13)'.
```

7.1 Analysis of the Matching Logic

7.1.1 Analysis of ISCNStudent>>matchSetTo:

```
matchSetTo: iscnExpr
        "Return an OC of closest matching mops that match to all
        the abnormalities in the iscnExpr."
    | aMatchMOP oldMop  abNodes acc removedAbNodes |

1. acc := ISCNExprAccessor new expr: iscnExpr.
```

The top level matching method, #matchSetTo: receives an ISCN expression as argument, and returns a set of existing cases that match to it for abnormality interpretation purposes. In reference 1, an expression accessor is created which parses the ISCN expression into a derivation tree, using the T-gen generated LR(1) parser. This object contains the knowledge to access the constituent grammatical components of the sentence.

```
2.     (aMatchMOP := self closestMatchTo: iscnExpr) isNil
              ifTrue: [^Array with: nil].
```

Reference 2 performs the essential match. The **#closestMatchTo**: method (Listing 7.3) returns a single case that matches best with respect to intersection sets of the abnormality branches of the derivation trees. Actually, a wrapper object which includes both the case, and the intersecting parse tree branches is returned. 'Best' in this sense is that there are common abnormalities expressed grammatically, and that it is the case with the least number of abnormalities. Why least as opposed to most? Experience with least vs. most strategies demonstrated that the later adaptation step was more simple and elegant using a large number of minimal abnormality matches rather than a small set of maximum abnormality matches. Using a small set of maximum abnormality matches led to an adaptation solution that was very awkward to program; the synthesis of abnormality descriptions from several cases was excessively complicated with respect to handling generalizations and definition of new specializations. In constrast, the synthesis of abnormality descriptions from several simple cases was based on a union of the abnormalities in a straightforward algorithm.

```
        "// match covers all abnormalities? "
3.     (aMatchMOP interSet size = acc abnormalities size)
              ifTrue: [^OrderedCollection with:
                      (aMatchMOP relatedAbNodes: acc abnormalityNodes)].
```

Reference 3 determines if the number of abnormalities in the expression is equal to the size of the intersection set of matching parse tree abnormality branches - implying the old case matched all the abnormalities. If it has, then we are finished, and the method returns the matching wrapper object of case, intersection set and explicit abnormality nodes in the derivation tree.

```
          "// isolate unmatch abnormalities and continue to
              use #closetMatchTo: on new (smaller) iscnExprs
              with the remaining unmatched abs, until
              all have been covered"

                 "// in the interSet are abs, so this
                    removes abs from the expr"
4.    removedAbNodes := acc removeNodes: aMatchMOP interSet.
```

In reference 4 the accessor object's knowledge of the grammatical structure of ISCN expressions is used to remove those abnormality nodes from the derivation tree that have been successfully matched on and for which existing cases have been found.

```
                 "// RECURSE for the remaining matches on a
                    smaller expr."
5.    ^(OrderedCollection with:
      (aMatchMOP relatedAbNodes: removedAbNodes))
           addAll: (self matchSetTo: acc expr);
           yourself.
```

In reference 5, the final step in the matching process, the essential recursive construction occurs. We return the **ISCNMatchingMOP** object (our wrapper of existing case and matching abnormality derivation tree branches) with the match set for the remaining expression, constructed via a recursive invocation of the method.

```
matchSetTo: iscnExpr
       "Return an OC of closest matching mops that match to all
       the abnormalities in the iscnExpr."
    | aMatchMOP oldMop  abNodes acc removedAbNodes |

    acc := ISCNExprAccessor new expr: iscnExpr.

    (aMatchMOP := self closestMatchTo: iscnExpr) isNil
            ifTrue: [^Array with: nil].
        "// match covers all abnormalities? "
    (aMatchMOP interSet size = acc abnormalities size)
            ifTrue: [^OrderedCollection with:
                    (aMatchMOP relatedAbNodes: acc abnormalityNodes)].

        "// isolate unmatch abnormalities and continue to
            use #closetMatchTo: on new (smaller) iscnExprs
            with the remaining unmatched abs, until
            all have been covered"

               "// in the interSet are abs, so this
                  removes abs from the expr"
    removedAbNodes := acc removeNodes: aMatchMOP interSet.
                "// RECURSE for the remaining matches on a
                  smaller expr."
    ^(OrderedCollection with:
      (aMatchMOP relatedAbNodes: removedAbNodes))
            addAll: (self matchSetTo: acc expr);
            yourself.
```

Listing 7.2 **The Top Level *Matching* Method for ISCN Student**

7.1.2 Analysis of ISCNStudent>>closestMatchTo:

```
closestMatchTo: iscnExpr
       "Return the closest matching case to the iscnExpr,
           based on intersections of 'abnormality' sections
           in the parse trees. Only consider generalized
           cases, not leafs."
    |  newCaseTree oldCaseTree coll interSet maxSize minSize
       exprAcc maxDepth |

           "// candidate matching cases are ordered by the
                 number of abnormalities they cover for the
                 new case."
 1.   coll := SortedCollection sortBlock: [:a :b |
        (a mop fillerFor: 'Abnormalities') groupSize <
        (b mop fillerFor: 'Abnormalities') groupSize].
```

The heart of the matching is done in **#closestMatchTo**: . The method finds and returns a single object of class **ISCNMatchingMOP**, which packages the matching case and associated matching information. The basis of matching is comparison of the abnormality nodes in the derivation trees. The case which has the set of abnormalities is chosen. Note that this case has associations to a detailed interpretation which can be used in constructing the interpretation of the new case.

In reference 1 a priority queue (SortedCollection) is defined that will contain the potential matches. The sorting criterion indicates preference to cases with the smallest number of abnormalities.

```
2.     exprAcc := ISCNExprAccessor new expr: iscnExpr.
3.     newCaseTree := exprAcc derivTree.
```

References 2 and 3 create an ISCN expression accessor, from which we extract a derivation tree for use in comparison.

```
              "// search through existing abstraction cases
                     for intersections of parse trees"
4.    ((self at: 'ISCNInterpretation') allSpecializations select: [:x |
5.        x isAbstraction and: [x specializations notEmpty]])
```

In references 4 and 5 we choose our candidates for comparison. Note that only abstraction cases that have specialized children are chosen. These abstractions are generalized cases in which information such as the specific chromosomes involved have been replaced with variables suitable in pattern matching.

```
6.          do: [:aMOP |
                oldCaseTree := aMOP fillerFor: 'DerivationTree'.
                interSet := oldCaseTree intersectionAt: 'abnorm'
                                        with: newCaseTree.
                interSet notEmpty ifTrue: [
                    coll add: (ISCNMatchingMOPDG new
                                      mop: aMOP;
                                      interSet: interSet;
                                      yourself)]].
```

In the section starting with reference 6, we iterate over all potential existing cases. The derivation tree for an existing case is retrieved. Then comes the critical step of finding intersection sets between the new and old derivation trees based on syntactic branches referring to abnormalities. Cases for which a match is found are packaged along with the resulting intersection set in a **ISCNMatchingMOP** object, and finally stored in the priority queue.

```
            "// select the case with the narrowest abnormality match"
7.      coll isEmpty
          ifTrue: [
            MessageBox message:
                'Failure: No old case matches to ' , iscnExpr ,
                '. Adaptation is not possible.'.
            ^nil]
          ifFalse: [
            ^coll first].
```

Reference 7 tests if any matching cases were indeed found. If there are, the priority queue ensures they are sorted in increasing order of abnormality matches. We choose and return the first match - the one with the *least* number of abnormalities matching the new case.

```
closestMatchTo: iscnExpr
        "Return the closest matching case to the iscnExpr,
            based on intersections of 'abnormality' sections
            in the parse trees. Only consider generalized
            cases, not leafs."
    | newCaseTree oldCaseTree coll interSet maxSize minSize
      exprAcc maxDepth |
        "// candidate matching cases are ordered by the
              number of abnormalities they cover for the
              new case."
coll := SortedCollection sortBlock: [:a :b |
    (a mop fillerFor: 'Abnormalities') groupSize <
    (b mop fillerFor: 'Abnormalities') groupSize].

exprAcc := ISCNExprAccessor new expr: iscnExpr.
newCaseTree := exprAcc derivTree.

        "// search through existing abstraction cases
              for intersections of parse trees"
((self at: 'ISCNInterpretation') allSpecializations select: [:x |
    x isAbstraction and: [x specializations notEmpty]])
        do: [:aMOP |
            oldCaseTree := aMOP fillerFor: 'DerivationTree'.
            interSet := oldCaseTree intersectionAt: 'abnorm'
                                        with: newCaseTree.
            interSet notEmpty ifTrue: [
                coll add: (ISCNMatchingMOPDG new
                                    mop: aMOP;
                                    interSet: interSet;
                                    yourself)]].

        "// select the case with the narrowest abnormality match"
coll isEmpty
    ifTrue: [
        MessageBox message:
            'Failure: No old case matches to ' , iscnExpr ,
            '. Adaptation is not possible.'.
        ^nil]
    ifFalse: [
        ^coll first].
```

Listing 7.3 **The Essential *Matching* Method that is Recursively Called**

8. Adaptation

Adaptation is the act of modifying the solution in an existing case (or cases) to apply to the new case. The case or cases chosen for adaptation are those which have been identified as most closely matching the new input case. Once constructed, the adapted solution is associated with the new case, which is then stored in case memory. Adaptation is typically the most complex, knowledge intensive and domain dependent reasoning process in CBR systems, as modification of a solution requires awareness of domain principles and heuristics.

In CBR theory, there are two main categories of adaptation: structural and derivational (Kolodner 87). Structural adaptation methods apply rules of modification directly to the retrieved (best) case. Derivation methods involves discovery of the rules that generated the retrieved case, and then applying these rules to generate a new solution. With either approach critic-based adaptation (Sacerdoti 1975; Hammond 1989) is a common addition, in which critics are used to recognize problems in nearly correct solutions; feedback may re-invoke the adaptation phase to converge on a better solution. ISCN Student uses a structural adaptation technique, without the use of a critic.

In the case of ISCN Student the solution is a semantic network describing the chromosomal abnormality features associated with an ISCN expression, such as the

MOPs describing abnormal chromosome copies. The adaptation process involves extracting existing solutions from the best matching cases and transforming the abnormality descriptions to fit the new case.

```
interpretNewCaseFor: iscnExpr
      | closestOld new matches case |

      (case := self at: iscnExpr) notNil ifTrue: [^case].
      matches := self matchSetTo: iscnExpr.
      (matches includes: nil) ifTrue: [^nil].

            "// construct a new case from the old matching ones."
    ^self adapt: matches forExpr: iscnExpr.
```

Listing 8.1 Top Level Method for Case Interpretation. Performs Matching and Adaptation

As previously reviewed, **ISCNStudent>>interpretNewCaseFor:** (Listing 8.1) is the primary method used to construct new case solutions. After the best matching existing cases have been found, the statement:

```
      ^self adapt: matches forExpr: iscnExpr.
```

invokes the **ISCNStudent>>adapt:forExpr:** method which constructs a new interpretation solution from the existing ones. It is installed in case memory as a side-effect.

8.1 Analysis of the Adaptation Logic

8.1.1 Analysis of ISCNStudent>> adapt:forExpr:

```
adapt: matches forExpr: iscnExpr
        "Return a new MOP case that is an adaptation of the related
        matching old cases. The adaptation is based on iscnExpr.
        Also install
        this new mop and all associated mops (such as abnormalities)"

  | x exprAccessor specAbExprs slots mopStruct mop chCopies longForms |

1.    exprAccessor := ISCNExprAccessor new expr: iscnExpr.
```

The top level adaptation method, **#adapt:forExpr:** receives as arguments:

- matches - a set of **ISCNMatchingMOP** objects. These are composite objects containing the matching case MOP and the intersection set of matching abnormalities.
- iscnExpr - a string ISCN expression.

In reference 1, an expression accessor is created which parses the ISCN expression into a derivation and abstract syntax tree.

```
      "// calc the adaptations for abnormalites, copies, etc"
2.  specAbExprs := self adaptAbnormalitesOf: matches forExpr: iscnExpr.
3.  chCopies := self adaptChCopiesOf: matches forExpr: iscnExpr.
4.  longForms := self adaptISCNLongFormsOf: matches forExpr: iscnExpr.
```

References 2-4 are the heart of the adaptation phase, constructing the adapted interpretations in the three central categories:

1. abnormality descriptions
2. abnormal chromosome copy counts

3. ISCN long form notation for each structural aberration

```
"// create the new case MOP and install it"
4.    mopStruct := OrderedCollection new.
      ...
      ...
      ...
      ...
      ^mop := self storeCollAsMOP: mopStruct.!
```

From reference 4 until the end of the method simple housekeeping processes transform the three adapted interpretation sections into a proper case MOP and install it in memory, finally returning the result.

```
adapt: matches forExpr: iscnExpr
        "Return a new MOP case that is an adaptation of the related
        matching old cases. The adaptation is based on iscnExpr.
        Also install
        this new mop and all associated mops (such as abnormalities)"

  | x exprAccessor specAbExprs slots mopStruct mop chCopies longForms |

    exprAccessor := ISCNExprAccessor new expr: iscnExpr.

        "// calc the adaptations for abnormalites, copies, etc"
    specAbExprs := self adaptAbnormalitesOf: matches forExpr: iscnExpr.
    chCopies := self adaptChCopiesOf: matches forExpr: iscnExpr.
    longForms := self adaptISCNLongFormsOf: matches forExpr: iscnExpr.

        "// create the new case MOP and install it"
    mopStruct := OrderedCollection new.
    slots := OrderedCollection new.
    slots
        add: (Array with: #Expression with: iscnExpr);
        add: (Array with: #ChCnt with: exprAccessor chCnt);
        add: (Array with: #Sex with: exprAccessor sexModel).

        "// add in the 'group' attributes like abs and chCopies"
    specAbExprs notEmpty ifTrue: [
            x := specAbExprs collect: [:aString | Array with: aString].
            slots add:(Array with: #Abnormalities with: x)].
    chCopies notEmpty ifTrue: [
            x := chCopies collect: [:aString | Array with: aString].
            slots add:(Array with: #ChCopies with: x)].
    longForms notEmpty ifTrue: [
            x := longForms collect: [:aString | Array with: aString].
            slots add:(Array with: #ISCNLongForms with: x)].

        "// put it all together in a mop and save it in memory"
    mopStruct
        add: iscnExpr;  "// name "
                "// generalizations"
        add: (matches collect: [:aMatchMOP | aMatchMOP mop name]);
        add: #instance;
        add: slots.
    ^mop := self storeCollAsMOP: mopStruct.!
```

Listing 8.2 **The Top Level *Adaptation* Method for ISCN Student**

8.1.2 Analysis of ISCNStudent>> adaptAbnormalitesOf:forExpr:

Three adaptation processes occur in ISCN Student: for 1) abnormalities, 2) chromosome copies, and 3) ISCN long form expressions. A detailed analysis of each process would be both overwhelming and tedious! Therefore only the adaptation of abnormalities is considered explicitly as representative of the domain knowledge required to construct new solutions from old. As suggested by the analysis of this first category, the processes required for adapting copies and long form expressions are also subtle and complex.

```
adaptAbnormalitesOf: matches forExpr: iscnExpr
        "Answer a coll of abnormalies for iscnExpr, adapted
        from the matching old cases. Install these abnormalies
        in memory too"
    | x exprAccessor genAbExprs genAbs specAbExprs
      genExpr specExpr abMOP mop slots |

1.    exprAccessor := ISCNExprAccessor new expr: iscnExpr.
        "// strip out the generalized ISCN abnormality expressions"
2.    genAbExprs := exprAccessor generalizedAbnormalities.
        "// strip out the specialized (with nums)
            ISCN abnormality expressions"
3.    specAbExprs := exprAccessor abnormalities.
```

Adaptation of abnormalities requires the new ISCN expression and best matching cases as arguments. In reference 1 an ISCN expression accessor object is created that constructs a parse tree and abstract syntax tree for the expression.

In reference 2 the accessor object uses its specialized knowledge of the abstract syntax tree to extract those portions of the expression that indicate structural abnormalities, generalized so that all chromosome references are made into variables.

For example, in the expression '46,xy,del(1)(p21)', a generalized expression of the form 'del(<x1>)(p<x2>)' would be returned.

At reference 3 the accessor returns an expression similar to that in reference 2 - the structural abnormalities. But where in reference 2 they were generalized with respect to chromosome numbers, here the numbers are kept. Again, domain knowledge of the derivation tree and nodes that comprise abnormalities is exploited.

Thus the expression '46,xy,del(1)(p21)' would return 'del(1)(p21)'.

```
"// collect the associated generalized abnormality mops. We
          can assume these are 'generalized abs' because the match
          set contains generalized cases, not specific ones."
     genAbs := OrderedCollection new.
 4.  (matches collect: [:aMatchMOP | aMatchMOP mop]) do: [:aMOP |
          genAbs addAll:
               (aMOP fillerFor: 'Abnormalities') groupMembers].
```

At reference 4 the MOPs representing the generalized abnormality descriptions associated with the best matching old cases are extracted. For example, if the case was '46,xy,del(5)(p11)', this would yield a generalized MOP for the '**terminal deletion**' abnormality.

```
"// for each general abnorm expr (e.g. +<num> ) in the iscnExpr,
          find the associated generalized abnormality. Then make the
          specific abnormality (+2) a specialization
          of the generalized
          one (extra whole...) ."
5.   1 to: genAbExprs size do: [:i |
```

Reference 5 frames a major iteration over each generalized abnormality expression, such as 'del(<x1>)(p<x2>)'. Within this loop, the associated generalized abnormality (e.g. '**terminal deletion**') will be found and matched with 'del(<x1>)(p<x2>)'. Then the specific abnormality expression associated with the general one (e.g. 'del(1)(p21)' with 'del(<x1>)(p<x2>)') is defined as a specialization of the generalized abnormality concept (e.g. '**terminal deletion**').

```
6.          genExpr := genAbExprs at: i.          "// e.g. +<num> "
7.          specExpr := specAbExprs at: i.      "// e.g. +2 "
                    "// find the generalized abnormality that matches to
                        the generalized expr.
                        e.g. 'extra whole...' with +<num> "
8.          abMOP := genAbs
                    detect: [:aMOP |
                        (aMOP fillerFor: 'ExprFormat') = genExpr]
                    ifNone: [self error: 'missing abnormality'].
```

Within the iteration process, reference 6 extracts the current generalized expression - for example, 'del(<x1>)(p<x2>)'.

Reference 7 extracts the associated specialized expression - for example, 'del(1)(p21)'.

Reference 8 searches for a generalized abnormality concept MOP whose generalized expression format (e.g. 'del(<x1>)(p<x2>)') matches the one currently sought. Once found, it is stored in the temporary 'abMOP' variable.

```
                  "// make the specific ab a specialization
                        of the general"
9.          slots := Array with: (Array with: #Code with: '???').
            self storeCollAsMOP:
                (Array
                              "// mop name. e.g. +2"
                     with: specExpr
                              "// abstraction. e.g. extra whole..."
```

```
                    with: (Array with: abMOP name)
                    with: #abstraction
                    with: slots) ].
```

Still within the iteration, and starting with reference 9, the new specific abnormality expression of the new case (e.g. 'del(1)(p21)') is stored in MOP memory as a specialization of the generalized abnormality concept MOP (e.g. **'terminal deletion'**). Thus the generalization-specialization hierarchy is extended during reasoning in a semantically meaningful way, with new abnormalities correctly inserted as specialization's of their appropriate generalized genetic defect concept.

The above references 6-9 iteratively repeat until all abnormalities in the new ISCN expression case have been constructed into MOPs and placed in the abnormality hierarchy.

```
10.    ^specAbExprs.
```

Finally, the set of specialized abnormality expressions for the new expression is returned to the calling method, for construction into the complete new case MOP and its associations to other MOPs.

```
adaptAbnormalitesOf: matches forExpr: iscnExpr
        "Answer a coll of abnormalies for iscnExpr, adapted
        from the matching old cases. Install these abnormalies
        in memory too"
    | x exprAccessor genAbExprs genAbs specAbExprs
      genExpr specExpr abMOP mop slots |

    exprAccessor := ISCNExprAccessor new expr: iscnExpr.
        "// strip out the generalized ISCN abnormality expressions"
    genAbExprs := exprAccessor generalizedAbnormalities.
        "// strip out the specialized (with nums) ISCN
            abnormality expressions"
    specAbExprs := exprAccessor abnormalities.
    "// collect the associated generalized abnormality mops. We
        can assume these are 'generalized abs' because the match
        set contains generalized cases, not specific ones."
genAbs := OrderedCollection new.
(matches collect: [:aMatchMOP | aMatchMOP mop]) do: [:aMOP |
        genAbs addAll:
              (aMOP fillerFor: 'Abnormalities') groupMembers].

    "// for each general abnorm expr (e.g. +<num> ) in the iscnExpr,
        find the associated generalized abnormality. Then make the
        specific abnormality (+2) a specialization
        of the generalized
        one (extra whole...) ."
1 to: genAbExprs size do: [:i |
        genExpr := genAbExprs at: i.          "// e.g. +<num> "
        specExpr := specAbExprs at: i.      "// e.g. +2 "
                "// find the generalized abnormality that matches to
                    the generalized expr.
                    e.g. 'extra whole...' with +<num> "
        abMOP := genAbs
                detect: [:aMOP |
                    (aMOP fillerFor: 'ExprFormat') = genExpr]
                ifNone: [self error: 'missing abnormality'].

                "// make the specific ab a specialization
                    of the general"
        slots := Array with: (Array with: #Code with: '???').
        self storeCollAsMOP:
            (Array
                        "// mop name. e.g. +2"
                with: specExpr
                        "// abstraction. e.g. extra whole..."
                with: (Array with: abMOP name)
                with: #abstraction
                with: slots) ].

^specAbExprs.
```

Listing 8.3 Adaptation of abnormalities

9. Visual Manipulation

9.1 Overview

The heart of ISCN Student is the capability to receive a new ISCN expression, as a character string, and interpret this new case in terms of existing cases using CBR techniques. For example:

```
        aCBRChrom interpretNewCaseFor: '46,xy,del(1)(p21)'.
```

ISCN expressions represent chromosomal abnormalities, which can also be pictured using a standard ideogram convention outlined in the definitive ISCN report (Harden 1985). For example, chromosome 4 is depicted as shown in Figure 9.1. Cytogeneticists, the domain experts who use ISCN notation, are comfortable communicating the defects using these ideograms with additional graphical annotation. However, there is no consensus on how abnormalities should be visually expressed.

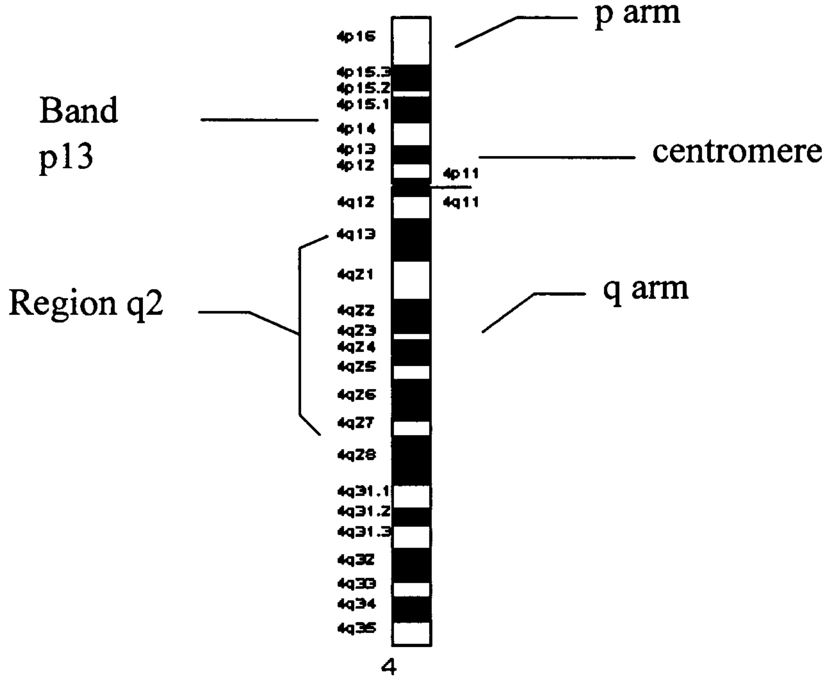


Figure 9.1 **The standardized ideogram for chromosome 4**

A chromosome ideogram with accompanying graphical annotation to indicate defects can be mapped to an ISCN expression. In ISCN Student, software has been developed for this mapping. The user introduces defects using a visual manipulation metaphor on the ideograms, such as pointing at and 'breaking off' the end of a chromosome ideogram. Each new visual modification leads to a diagram of the chromosomes that can be associated with a new ISCN expression. Once the new expression has been created, the conventional CBR mechanism for case interpretation for a character string format expression (**CBRChrom>>interpretNewCaseFor:**) can be invoked, yielding a new case.

In short, ISCN Student provides two mechanisms for entering new cases: 1) direct ISCN expression as a character string, or 2) indirect creation of an ISCN expression via visual manipulation of chromosome ideograms. In either case, the final CBR new case interpretation is done from the character string expression.

Four major components are needed to provide the capabilities of visual manipulation:

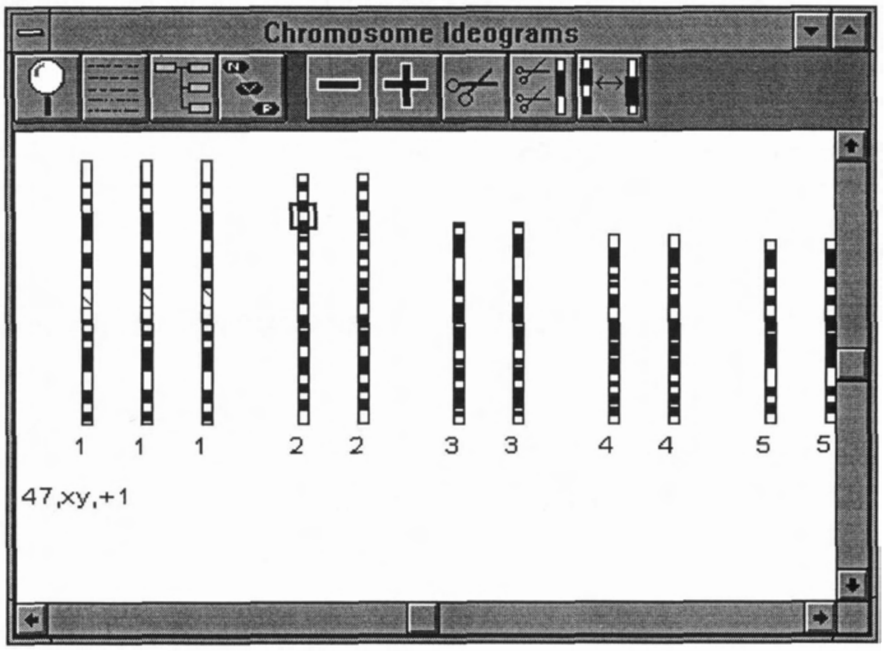
1. A correct object model that provides meaningful composition and semantic definitions for chromosome structures. From this, drawable ideograms can be generated.
2. A means to generate new chromosome topological structures for abnormalities.
3. A method to draw the ideograms from the chromosome topological structures.
4. A metaphor for visually manipulating the displayed ideograms.

The class which defines the responsibility for creating chromosome structures, both for normal and abnormal chromosomes, is **ChromCBR**. The class which defines responsibility for the display of chromosome ideograms and their visual manipulation is **ChromView**.

9.2 The Manipulation Metaphor

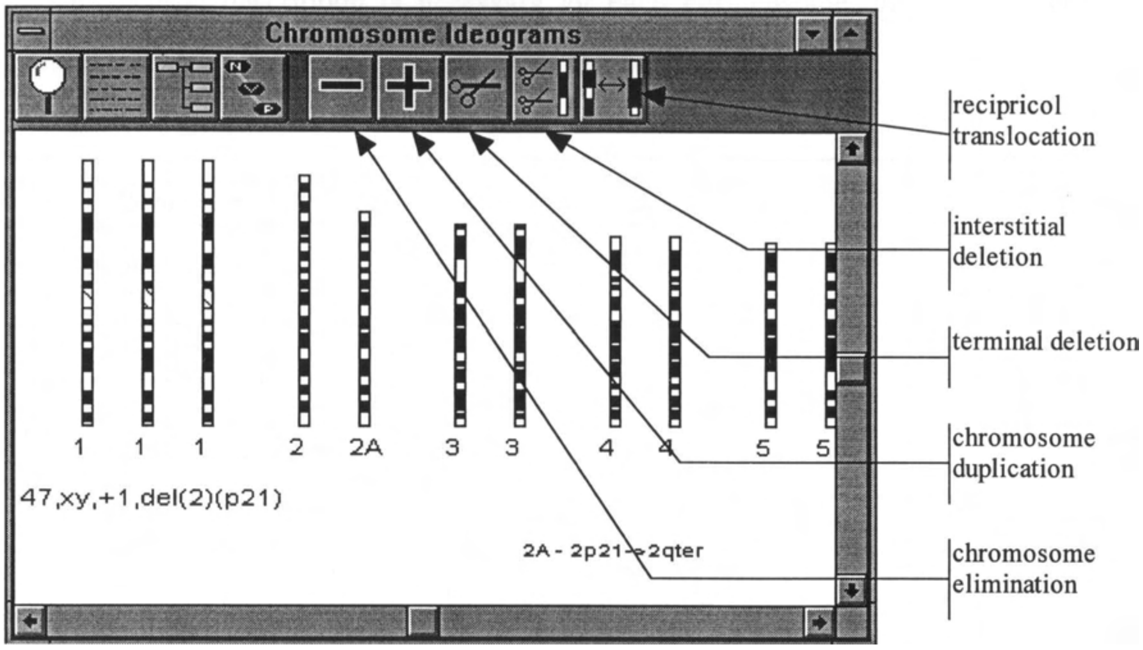
In the absence of any standards for graphical annotation of chromosome abnormalities on the ideograms, I was free to construct a metaphor for visually introducing new defects. The metaphor chosen was one of direct manipulation of structure, as all the defects are structural. For example, introducing a terminal deletion is accomplished by pointing to a chromosome band for deletion, and then pressing the *scissors* iconic button to effect the cut, or deletion (Figure 9.2 and Figure 9.3).

Thus terminal and interstitial deletions find obvious analogues in the scissors buttons, while translocations are suggested by the visual *swap* icon. The chromosome addition and elimination manipulation symbolism required a compromise from the physical iconic hints, underscoring the need to use the most familiar symbols in the domain of the user. In these cases, the '+' and '-' symbols were chosen, as these are used in ISCN to denote the abnormalities, and are also rich in associative meaning from arithmetic. Although more *physical* symbols are possible, they would have a lower associative strength to the intended action than the familiar and terse ISCN nomenclature.



47,xy,+1

Figure 9.2 **Visual Manipulation: Choosing a breakpoint on chromosome 2.**



47,xy,+1,del(2)(p21)

2A - 2p21->2qter

Figure 9.3 Visual Manipulation: Choosing the "Scissors" Button creates a terminal deletion

9.3 The Display of Ideograms for a Chromosome Model

long form in the case-> ideogram display

rep for ISCNInterpretation w.r.t. calc filler for chromosomes

9.3.1 The Human Chromosomes Object Model Used in Ideogram Drawings

Objects of class **ChromView** are responsible for drawing the chromosome ideograms associated with a case. In order to draw the ideograms for all chromosomes in a human karyotype, each case must be associated with a description for all chromosomes. A band by band topological description is necessary for each chromosome, including height, color, and if its terminal or centromeric (next to the centromere).

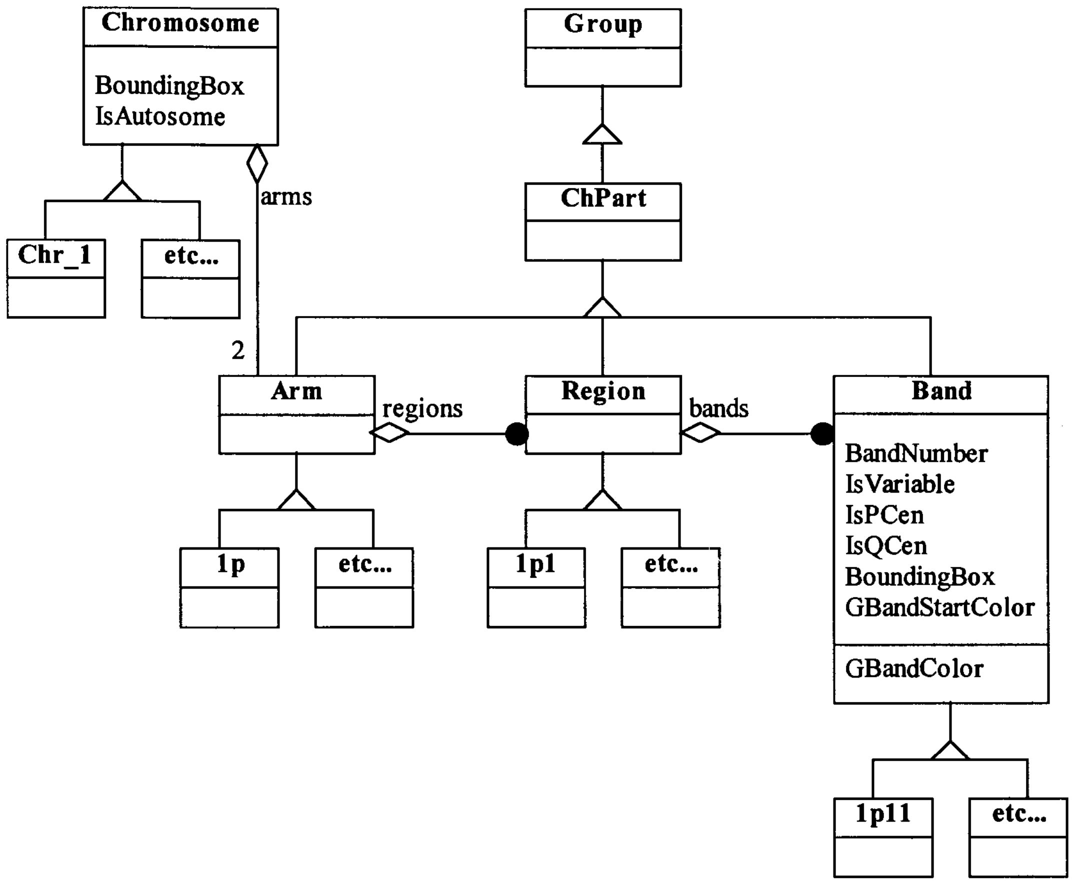


Figure 9.4 **Object model for human chromosomes and ideogram information**

In order to realize this goal, ISCN Student defines an object model for the human chromosomes (Listing 9.1 and Figure 9.4). Note that an aggregation model (indicated by the diamond in OMT notation) has been chosen to reflect the genetic notion of higher order structural units being composites of lower order units. For example, arms are defined by an aggregate ordered group of regions. Thus, chromosomes are composed of 2 arms which in turn are composed of many regions which are composed of many bands.

Specific chromosomes, arms, etc., are defined as subclasses of the abstract superclasses and these are inserted into the generalization hierarchy of the MOP memory. These chromosome MOP structures are then linked via associative slots to the cases.

The preceding description pertains to *pristine* chromosomes; those without defect. The construction of *abnormal* chromosome MOP structures for use in ideogram displays, that is, those chromosomes related to aberration descriptions in the ISCN expression, are a more difficult case. Their creation is discussed in the next section.

In Listing 9.2 we see the detailed descriptions for the band layouts required for a chromosome (in this case, chromosome 1) in order to be able to display its ideogram. Height, color, position and other attributes are given for each band, and the bands are then aggregated into regions which in turn are aggregated into arms, and finally into complete a chromosome.

```
(Chromosome (Root)
        (   (Drawing (Pattern) (CalcFN drawCh:))
             (BoundingBox nil)
             (IsAutosome false)
             (Bands (Pattern) (CalcFN bandsFor:))
        )  )

(ChPart (Group))

(Arm (ChPart))

(Region (ChPart))

(Band (ChPart) abstraction
        (    (IsVariable false)
             (IsPCen false)
             (IsQCen false)
             (BoundingBox nil)
             (GBandColor (Pattern) (CalcFN gBandColor:))
             (Arm (Pattern) (CalcFN bandArm:))
             (BandNumber (Pattern) (CalcFN bandNumber:))
             (ChNumber (Pattern) (CalcFN bandChNumber:))
             (GBandStartColor nil)
        )  )
```

Listing 9.1 MOP definitions for chromosome object model

```
addCh1MOP
    "Add MOPs for a normal ch 1"

        "// 1p1 BANDS -------------------------------------------"
MOPParserDG
defMOP: #('1p11' (Band)
        (   (Height 1)
            (IsPCen true)
          )  ) in: self;

  defMOP:  #('1p12' (Band)
        (   (Height 1)
            (GBandStartColor ClrBlue)
          )  ) in: self;

  defMOP:  #('1p13' (Band)
        (   (Height 6)
         )  ) in: self;

      ...
      ...

      "// 1p REGIONS -------------------------------------------"
  defMOP:  #('1p1' (Region)
        (   (1 ('1p11'))
            (2 ('1p12'))
            (3 ('1p13'))
         )  ) in: self;

      ...
      ...

        "// 1p ARM -------------------------------------------"
  defMOP:  #('1p' (Arm)
        (   (1 ('1p1'))
            (2 ('1p2'))
            (3 ('1p3'))
         )  ) in: self;

      ...
      ...

        "// CHROMOSOME 1 -------------------------------------------"
   defMOP:  #(Ch1 (Chromosome)
    (    (Number 1)
         (IsAutosome true)
         (pArm ('1p'))
         (qArm ('1q'))
     )   ) in: self.
```

Listing 9.2 Sample abbreviated graphical ideogram band details for chromosome 1

9.3.2 The Generation of Chromosome MOPs for a Case

The generation of chromosome structures from which ideograms could be drawn turned out to be one of the most subtle problems in this work. The heart of the problem is that we start only with an ISCN expression, and from this must determine the precise topological structure of each abnormal chromosome, band by band.

As indicated in the last section, the topological and graphical descriptions necessary to create the ideograms for all pristine chromosomes are stored into the original state of MOP memory. These descriptions are not suitable for *abnormal* chromosomes, which by definition have structural defects. How, then, are the abnormal chromosome descriptions generated?

Each case MOP has associated with it a MOP filler named **Chromosome**, which is related to a slot demon that fires when the attribute is accessed (Listing 9.3). This causes the **CBRChrom>>chromosomes:** method to execute, which calculates and returns the set of both pristine and abnormal chromosomes (Listing 9.4).

```
(ISCNInterpretation (Case) abstraction
        (    (Expression nil)
             ...
             (Chromosomes (Pattern) (CalcFN chromosomes:))
             (AbnormalChs (Pattern) (CalcFN abnormalChs:))
             (SexChs (Pattern)      (CalcFN sexChs:))
             ...
     )   )
```

Listing 9.3 ISCNInterpreation case MOPs have Chromosome demons which calculate the chromosome structures.

Determining the pristine chromosome structures is trivial; they have been pre-defined. The abnormal chromosome construction requires more work - based on the ISCN long form expressions associated with a case. As suggested in previous sections on case adaptation, the long form expressions for new cases are adapted from existing best matching cases during the CBR adaptation phase.

To review, the long form expressions are part of the ISCN nomenclature that symbolically describe the contiguous structure of each defective chromosome. For example,

del(1)(q21q31) *is equivalent to* ***1pter→1q21::1q31→1qter***

The long form expression ***1pter→1q21::1q31→1qter*** provides a complete topological description at the gross level. Thus chromosome MOPs composed of arms, regions and bands as described in the last section can be constructed from this description.

Within the **#chromosomes:** method, the generation of copies of the *pristine* chromosome MOPs occurs in the section starting with reference 1 in Listing 9.4. There are no surprises here; just a need to ensure a very deep copy down to the band MOP level within the composite MOP object.

The generation of abnormal chromosome MOPs starts at reference 2. The key statement at reference 2 is:

```
(mop fillerFor: 'AbnormalChs') , (mop fillerFor: 'SexChs')
```

In this expression the chromosome MOPs for abnormal and sex chromosomes are generated as lists which are concatenated together. The following statements create deep copies of the list elements. As this statement suggests, the case MOP has further attribute demons for the abnormal and sex chromosomes (e.g. (mop fillerFor: 'AbnormalChs')). Through a series of indirect MOP demons, the processing will finally arrive at the heart of the logic, **#chForLongExpr:abnormalityOf:** and its helper, **#bandsFrom:to:** (Listing 9.5 and Listing 9.6).

```
chromosomes: aColl
        "MOP Calculation Function.
        Return the OC of chromosomes for this interpretation"
    | pattern mop chs copy |

    pattern := aColl at: 1.     mop := aColl at: 2.
    (mop isSuccessorOf: 'ISCNInterpretation') isSatisfied.

    chs := SortedCollection sortBlock: [:a :b |
            (a fillerFor: 'Number') asString  <
            (b fillerFor: 'Number') asString].

        "// normally, 2 of each autosome,
            unless a numerical abnormality"
1.    self autosomes do: [:aMOP |
                ... // deep copy logic for pristine chromosomes
                ...

        "// add in the abnormalites and sex"
2.    (mop fillerFor: 'AbnormalChs') ,  (mop fillerFor: 'SexChs')
            do: [:aMOP |
                copy := aMOP deepCopy.
                copy replaceFiller:  (copy fillerFor: 'Bands')
                        deepCopy forRole: 'Bands'.
                chs add: copy].

    ^chs asOrderedCollection.
```

Listing 9.4 **The Chromosome demon generates detailed structures for both pristine and abnormal chromosomes.**

At reference 1 in **#chForLongExpr:abnormalityOf:**, we create an ISCNLongExprAccessor object to parse out the band sections within a long form expression. For example, in the expression:

1pter→1q21::1q31→1qter

The band sections are:

1pter→1q21

1q31→1qter

Starting at reference 2 in Listing 9.5, we iterate over each band section, and for each one, use the helper method **#bandsFrom:to:** to generate the contiguous set of bands associated with the band section range, saving the results in a growing list.

```
chForLongExpr: longExpr abnormalityOf: chNum
        "Return a ch mop that contains the bands in the range defined
        in longExpr. This new ch mop will be an abnormal one, and
        will be installed in memory before it is returned. Its name will
        be longExpr. Its number will be chNum, if chNum is
        nonNil, else number will be first ch number encountered in
        longExpr."
    | acc bands slots num |

      bands := OrderedCollection new.
1.    acc := ISCNLongExprAccessorDG new expr: longExpr.

2.    acc bandSections do: [:aNode |
            bands addAll: (self bandsFrom: (acc startBandFor: aNode)
                                       to: (acc endBandFor: aNode))].

      ... // wrap up and return the bands as a Chromosome MOP
      ...
```

Listing 9.5 **Method to generate chromosome band structure for abnormal chromosomes, based on the ISCN long form expression.**

At references 1 and 2 in **#bandsFrom:to:** (Listing 9.6) we translate the 'familiar' band names as used in long form expressions, such as **1pter**, into their related band names, such as **1p36**. In reference 3 we iterate over the set of existing **Band** MOPs, and retrieve those that fall between the start and end bands in the specified range. Finally, in reference 4, the bands are ordered according to chromosome topological rules.

```
bandsFrom: start to: end
      "Return an OC of bands in the range from start to end"
    | bands startName endName |

         "// if start/end are a 'ter' or 'cen', get
             the associated band name.
             e.g. 1pter == 1p36 "
1.    startName := self bandNameEquivalentTo: start.
2.    endName := self bandNameEquivalentTo: end.
3.    bands := (self at: 'Band') specializations select: [:aMOP |
         self isBand: aMOP name between: startName and: endName].
4.   ^self sortedBands: bands from: startName to: endName.
```

Listing 9.6 Method used to generate chromosome bands in a range.

9.3.3 The Drawing of Chromosome MOPs

The drawing of the ideograms is a straightforward process. The hard work of determining the band structure for each chromosome in the case to displayed has already been accomplished, as reviewed in the previous section. The drawing algorithms simply traverse the contiguous bands that comprise the chromosome MOPs (and which contain detailed graphical height and color information) and draw each band.

In Smalltalk, the opening of a window has associated with it an event handler for drawing the window's client area. The window defined for drawing ideograms is defined in the **ISCNView** class, and the event handler for drawing is **#onGetContents**: (Listing 9.7). At reference 1 we iterate over all chromosome structures associated with the case. These are the topological structures for pristine and abnormal chromosomes. Within this iteration, we invoke the helper method **#draw:on:** (reference 2) to draw a particular chromosome.

```
onGetContents: aPane
      "Private. Draw the chrom on the pane"
    | prevCh pen extraGap bigGap basicGap |
      ... // initialization and label drawing
      ...

        "// draw each ch"
1. self chroms do: [:ch |

        ... // calculation of the ideogram position
        ...
2.      self draw: ch on: aPane.
      ].

    CursorManager normal change.
```

Listing 9.7 The top level method for drawing all chromosome ideograms.

In Listing 9.8 we see the essential process for drawing an ideogram. The bands for the chromosome MOP are extracted (reference 1) and we iterate over them in contiguous order. At reference 2 the detailed graphical drawing logic, based in part on the height of the band, is deferred to the helper method **#drawBand:on:at:** .

```
draw: ch on: aPane
        "Private. Draw the ch on the pane."
            | regions pt pen h bands pCen qCen b |

1.    bands := ch fillerFor: 'Bands'.
      ... // initialization; placing drawing pen at correct position
      ...

        "// draw the bands"
      pen drawRetainPicture: [
        bands do: [:band |
 2.         self drawBand: band on: aPane at: pt.
            h := self scaledHeight: (band fillerFor: 'Height').
            pt := pt up: h.
            ]].

      ... // draw the name and legend for this chromosome
      ...
```

Listing 9.8 Method for drawing an ideogram for a single chromosome.

9.4 Mapping Visual Changes to New Expressions

How does the visual manipulation which introduces abnormalities result in new ISCN expressions with deep interpretations, which in turn are redrawn correctly? When a manipulation icon is pressed, such as the *scissors* icon for terminal deletion, a button event handler method executes. The event handler has knowledge of the required ISCN syntactic form related to the type of modification. It identifies the structural change selected on the graphical representation, and then uses an **ISCNExprAccessor** object to append the appropriate ISCN syntactic unit for the abnormality to the ISCN expression associated with the ideograms. Finally, the new ISCN expression is evaluated as a new case using **CBRChrom>>interpretNewCaseFor:** (the main CBR match and adapt process). If the case is interpretable, a new deep model is constructed as discussed in previous sections. Associated with this new case will be the just-created ISCN long form expressions for each abnormal chromosome. As discussed above, from this set of long form expressions, a new set of chromosome MOPs with correct topological structure will be created (at considerable effort!), and finally the display of ideograms will be redrawn to reflect the structure of each pristine and defective chromosome.

```
deleteTerminal
        "Delete from the selected band to terminal"
    | band |

1.    (band := self selections at: #selectedBand ifAbsent: [nil])
                 isNil ifTrue: [^nil].

        "// mod the iscn expr"
2.    self exprAcc addDeleteTerminal: band.
3.    self refreshExpr.

        "// refresh window"
4.    self changed: #onGetContents:.
```

```
refreshExpr
        "Regenerate the iscn expr from the parse tree. This is
        invoked after the tree is modified. This is a major step
        because resetting the expr causes a new case to be
        retrieved or created (i.e. adaptation)"

        "// setting the expr causing new case and chromosomes
            to be generated"
5.    self expr: self exprAcc expr.



expr: aString
        "Set the value of the iscn expr (e.g. 46,xy) and
        all associated attributes (e.g. caseMOP, chromosomes)."
    | newCase |

          "// will return existing case, if one exists, or
              new adapted case."
6.     newCase :=self mopMem interpretNewCaseFor: aString.

          "// if no possible new case, return the expr accessor to
              its original expr and bail out."
7.     newCase isNil ifTrue: [
            self exprAcc: (ISCNExprAccessor new expr: self expr).
            ^nil].

8.     self case: newCase.
9.     expr := aString.
10.    self exprAcc: (ISCNExprAccessor new expr: expr).
11.    self chroms removeAll.
12.    self chroms addAll: (self case fillerFor: 'Chromosomes').
```

Listing 9.9 Translating a terminal deletion visual manipulation into a new case

As show in Listing 9.9, on pressing the scissors button to perform a terminal deletion, the #deleteTerminal method executes. The following processes then occur:

Reference	Explanation
1	The breakpoint band that should have previously been chosen is

retrieved.

2 The **ISCNExprAccessor** object has syntactic knowledge of ISCN and the #addTerminalDeletion message is sent with the breakpoint band in order to append the appropriate syntactic unit to the expression; for example, 'del(2)(p21)'.

3 The major step of refreshing the expression and associated interpretation model. Discussed in references 5-12.

4 Cause the graphical display to refresh and draw the new ideograms for the modified ISCN expression.

5 Extract the new expression from the **ISCNExprAccessor** object and assign it to the visual view's expression. This will initiate the CBR reinterpretation process (references 6-12).

6 The central CBR step of search and adaptation using #interpretNewCaseFor: executes with the new ISCN expression. A new case is generated and returned.

7 Failure handling if no new case can be constructed.

8-12 Record the new case and retrieve the chromosome structures associated with it. These structures will be used in the visual display processes and are described in the previous section on the ideogram drawings.

10. Evaluation

ISCN Student is a CBR system designed to learn - from ISCN Expert - how to interpret ISCN expressions. To evaluate how well it meets the goal of correctly interpreting new expressions, the interpretations of ISCN Expert are used as a standard for comparison. ISCN Expert was extensively tested on over 300 complex expressions, and scrutinized for accuracy by an expert in genetics and ISCN. Thus its results can be considered a reliable benchmark.

ISCN Student was also evaluated on performance; results will be presented.

10.1 ISCN Student Performance

Two performance metrics were gathered. The results show ISCN Student performed more poorly than ISCN Expert, and performance degraded as case memory increased. The tests were:

1. Relative performance of ISCN Student to ISCN Expert. For a simple case involving only one new abnormality (with respect to the case memory), what was the speed performance of ISCN Student compared to ISCN Expert? The test expression was:

47,xy,+2,del(1)(p21)

	Milliseconds
ISCN Student	1930
ISCN Expert	970

The results show that Expert is significantly faster (darn!). Possible causes include:

- Expert is in compiled machine code, Student is in interpreted Smalltalk.
- Student is performing more work due in the matching and adaptation cycles than Expert is in its rule-based backward chaining with backtracking.

2. Relative performance of ISCN Student as it learns. What was the time required to interpret a complex expression starting from a simple memory versus the time required with a larger memory containing more cases that closely match the new expression? The test expression was:

47,xy,+2,del(1)(p21),t(2;3)(q13;p21)

	Milliseconds
Simple case memory	4000
Rich case memory	5880

The results show that Student degraded in performance as memory grew (double darn!). Possible causes include:

- Increased search time during the matching phase, as case memory is larger.

- Adaptation phase logic performs better with construction from an aggregation of simple cases rather than complex ones.

10.2 ISCN Student Correctness

Two evaluations of expressions were generated. In both cases ISCN Student performed perfectly, generating identical interpretations to that of ISCN Expert.

The evaluation method for correctness was as follows: A set of ISCN expressions was loaded into ISCN Student to form the basis of the CBR memory (Listing 10.1). New expressions were then applied to ISCN Student that should be interpretable using the CBR learning techniques of matching and adaptation. The ISCN Student interpretations for these were compared to interpretations from ISCN Expert for the same expressions.

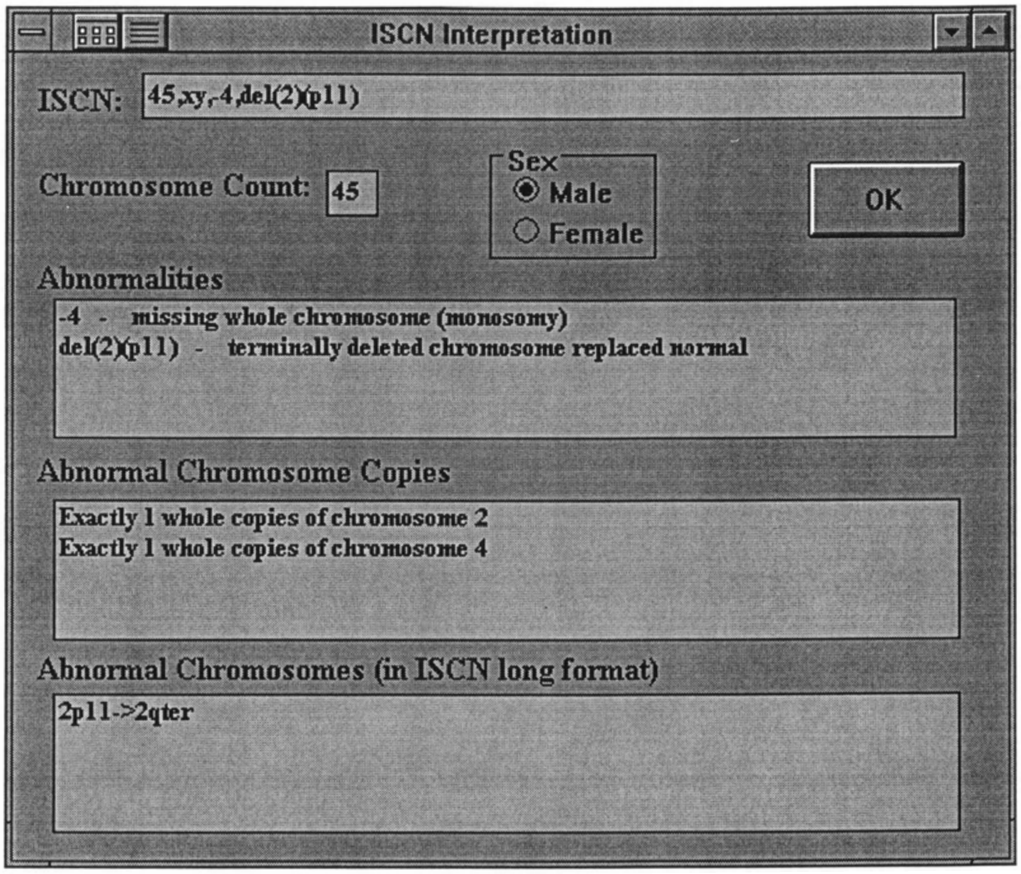
```
46,xy
46,xx
47,xy,+2
47,xx,+3
45,xy,-2
45,xx,-3
46,xy,del(1)(p21)
46,xy,t(2;3)(q13;p21)
46,xy,del(1)(p21),t(2;3)(q13;p21)
```

Listing 10.1 Starting cases used in evaluations

The first (relatively simple) test expression was:

45,xy,-4,del(2)(p11)

Compared to the debacle in the performance evaluation, Student has fared well (Figure 10.1) in relation to the interpretation from Expert (Listing 10.2). On all points, Student has generated a complete and correct interpretation that matches the results created by Expert.



ISCN: 45,xy,-4,del(2)(p11)

Chromosome Count: 45

Sex: [✓ Male] Female

Abnormalities

-4 - missing whole chromosome (monosomy)
del(2)(p11) - terminally deleted chromosome replaced normal

Abnormal Chromosome Copies

Exactly 1 whole copies of chromosome 2
Exactly 1 whole copies of chromosome 4

Abnormal Chromosomes (in ISCN long format)

2p11->2qter

Figure 10.1 ISCN Student interpretation for 45,xy,-4,del(2)(p11)

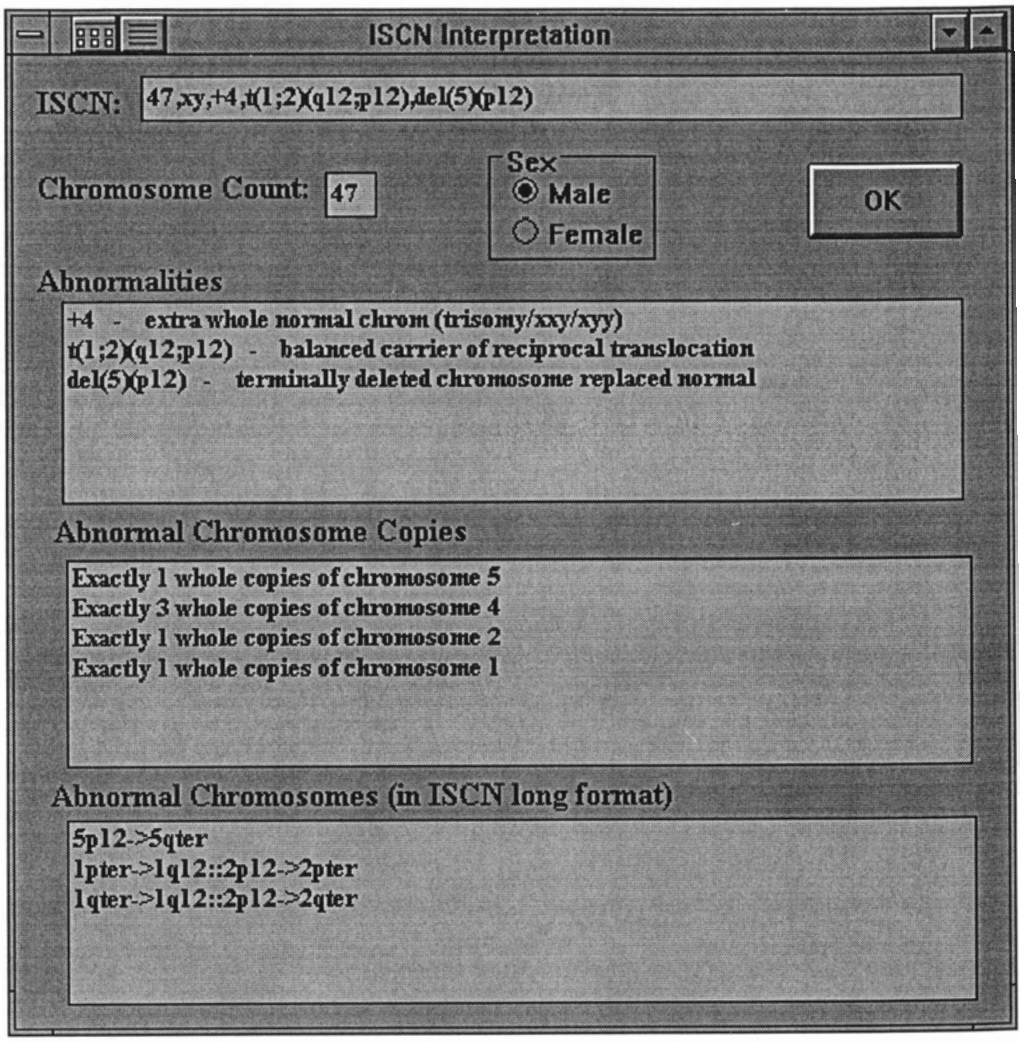
```
45,xy,-4,del(2)(p11)

Cell observation is terminal deletion
Chromosome 2 is 2p11->2qter
Sex model is male
Exactly 1 whole copies of chromosome 2
Exactly 1 whole copies of chromosome 4
TDR(2) = terminally deleted chromosome replaced normal
MIM(4) = missing whole chromosome (monosomy)
```

Listing 10.2 ISCN Expert interpretation for 45,xy,-4,del(2)(p11)

The second, more complex, test expression was:

47,xy,+4,t(1;2)(q12;p12),del(5)(p12)

Once again, Student's interpretation matches that produced by Expert (Figure 10.2 and Listing 10.3).



ISCN: 47,xy,+4,t(1;2)(q12;p12),del(5)(p12)

Chromosome Count: 47

Sex
[✓ Male]
Female

Abnormalities

+4 - extra whole normal chrom (trisomy/xxy/xyy)
t(1;2)(q12;p12) - balanced carrier of reciprocal translocation
del(5)(p12) - terminally deleted chromosome replaced normal

Abnormal Chromosome Copies

Exactly 1 whole copies of chromosome 5
Exactly 3 whole copies of chromosome 4
Exactly 1 whole copies of chromosome 2
Exactly 1 whole copies of chromosome 1

Abnormal Chromosomes (in ISCN long format)

5p12->5qter
1pter->1q12::2p12->2pter
1qter->1q12::2p12->2qter

Figure 10.2 ISCN Student interpretation for 47,xy,+4,t(1;2)(q12;p12),del(5)(p12)

```
47,xy,+4,t(1;2)(q12;p12),del(5)(p12)

Cell observation is reciprocal translocation
Cell observation is terminal deletion
Chromosome 1 is 1pter->1q12::2p12->2pter
Chromosome 2 is 1qter->1q12::2p12->2qter
Chromosome 5 is 5p12->5qter
Sex model is male
Exactly 1 whole copies of chromosome 1
Exactly 1 whole copies of chromosome 2
Exactly 1 whole copies of chromosome 5
Exactly 3 whole copies of chromosome 4
RTB(1,2) = balanced carrier of reciprocal translocation
TDR(5) = terminally deleted chromosome replaced normal
XNX(4) = extra whole normal chrom (trisomy/xxy/xyy)
```

Listing 10.3 ISCN Expert interpretation for 47,xy,+4,t(1;2)(q12;p12),del(5)(p12)

11.

Concluding Remarks

11.1 Contributions

The most significant contribution developed in the research for ISCN Student is the novel demonstration that a second generation knowledge system can automatically learn from a first generation predecessor, and achieve similar competence, using only CBR techniques for domain knowledge acquisition and reasoning. The input cases for developing a case memory are required to be the output solutions from the predecessor.

The first general purpose object-oriented framework for CBR systems was developed. It contains generic reasoning and representation support for all specialized CBR applications, and specialization hierarchies can be derived from it to create specific CBR systems.

The first formal grammar for ISCN was constructed and shown to be context-free for the subset of ISCN under consideration, the proof being the successful generation of an LR(1) parser for the grammar.

The novel integration of a visual manipulation front-end for chromosome ideograms used to generate symbolic problem cases for a CBR system demonstrated both the capability and convenience of such an approach.

11.2 Conclusions

The creation of ISCN Student was interesting and pleasurable (probably because it worked), with exploration into quite varied topics. CBR was verified as a sufficiently powerful technique to create a second generation knowledge system that learns from existing knowledge systems in order to achieve similar competence. This is the most novel research aspect of the thesis -- the demonstration of a CBR system that learns from other knowledge systems.

A generalized object-oriented framework and class hierarchy was designed for CBR systems that can be specialized into different CBR domains. The proof of its successful application is ISCN Student itself. A discovery in this work is that the very dynamic aspect of MOP memory (with MOPs being added, generalized and rearranged constantly) calls for a single class of MOP object. This MOP object itself takes on the dynamic roles of all instances and abstractions, with shifting attributes and behaviors. This is in contrast to the classic object-oriented design of pre-defining static class definitions for all anticipated abstractions. A regular class hierarchy in Smalltalk or C++ is not sufficiently dynamic or flexible for the demands of CBR reasoning and knowledge representation.

The crafting of the case memory representation and the loading of ISCN Expert interpretations led some insights. The choice of granularity of concepts, and what concepts to generalize, plays a pivotal role in the complexity of the matching and adaptation algorithms.

In considering the complexity of all software operations, the adaptation phase of the CBR system proved to be most intricate, and required the most domain knowledge. I believe this to be generally true of CBR systems, which underlines the need for further research into declarative and elegant reasoning and representation models for adaptation knowledge.

ISCN Student was coupled with a visual manipulation system for the display of existing (chromosome abnormality) cases and the introduction of new ones. Visual manipulation has both an intuitive appeal and an elegance of expression in this domain, which is largely concerned with topological defects.

Finally, I conclude that the CBR paradigm is worthy of more wide-spread exploitation, both in research and industry. The simplicity of the model, the often abundant availability of existing case histories, and its parallels to human cognitive problem solving strategies contribute to my feeling that CBR techniques have a practical appeal and great potential for successful application.

11.3 Future Research

As the evaluation section indicated, ISCN Student's performance degraded as the case memory grew. Analysis of the complexity of the matching and adaptation algorithms would be helpful to provide insight to the pace of degradation, and the source of potential improvements. Is it the matching or adaptation phase, or both, that is responsible for the majority of problem? Could the adaptation phase, which at present is optimized to work with simple cases, be accelerated if it worked better with complex cases?

The current naive drawing algorithm for chromosome ideograms is slow. Research into speeding it to the point where all 46 chromosomes could be drawn, instead of the representative 12, would improve the visual manipulation module.

ISCN Student has potential as an educational tool. An obvious area of inquiry, in this case, is identifying the cogent operational knowledge required by aspiring geneticists and the cognitive processes involved in the learning domain of ISCN and genetic abnormalities. These factors would assist the definition of new requirements for ISCN Student improvements, and could also be used in the creation of a computer-based student learning model within ISCN Student.

The generalization and adaptation techniques emphasize generalization of the specific chromosomes involved in defects. Further research into other dimensions of generalization of ISCN interpretation cases could yield new methods for matching and adaptation. The assistance of a domain expert would be helpful in the elucidation of other generalization attributes.

The adaptation phase relies heavily on domain knowledge. In ISCN Student this knowledge was embedded in the Smalltalk methods. There is room for improvement by investigating how to represent it declaratively, and apply it with a separate inference engine.

If ISCN Student fails in finding sufficient matches, it reports failure. An alternative would be to invoke ISCN Expert in these cases, and then dynamically load the new interpretation into case memory. This would make ISCN Student more robust, and accelerate its learning process.

ISCN Expert could be incorporated as an expert critic, either dynamically cross-checking Student's results, or doing so in a batch-mode when Student was not in use. Error detection and error handling research is called for to identify how wrong interpretations can be removed, and the associations in MOP memory adjusted to prevent the reoccurrence of incorrect results.

12. References

Booch, G. 1994. *Object-oriented Analysis and Design.* Redwood City, Ca: Benjamin-Cummings.

Chapman, S. 1989. *LR Parsing: Theory and Practice.* Cambridge, UK: Cambridge University Press.

Cooper, G. and Friedman, J. 1990. Interpreting Chromosomal Abnormalities Using Prolog, In *Computers and Biomedical Research,* 23:153-164. New York, NY: Academic Press.

Friedman, J. and Smith, J. 1986. Automated Interpretation of Cytogenetic Nomenclature, In *AAMSI Congress 86: Proceedings of the Congress on Medical Informatics.*

Goodman, M. 1989. CBR in Battle Planning. In *Proceedings: Workshop on case-based reasoning,* ed. K. Hammond. San Mateo, Ca: Morgan Kaufman.

Hagen, C. and Muller, J. 1993. *Focus on Scientific Visualization.* Berlin: Springer-Verlag.

Hammond, K. 1989. *Case-based Planning: Viewing Planning as a Memory Task.* Boston, MA: Academic Press.

Harden, D. and Klinger, H. 1985. *ISCN 1985: An International System for Human Cytogentic Nomenclature.* Basel, Switzerland: S. Karger AG.

Kolodner, J. and Riesbeck, C. 1986. *Experience, Memory and Reasoning.* Hillsdale, NJ: Lawrence Erlbaum.

Kolodner, J. 1987. Capitalizing on failure through case-based inference. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society.* Northvale, NJ: Erlbaum.

Kolodner, J. 1993. *Case-Based Reasoning.* San Mateo, CA: Morgan Kaufman.

Koton, P. 1988. Reasoning about evidence in causal explanation. In *Proceeding of AAAI-88*, pages 256-261. Los Altos, CA: Morgan Kaufman.

Long, W.; Naimi, S.; Criscitiello, M. and Jayes, R. 1987. The development and use of a causal model for reasoning about heart failure. In Symposium on Computer Applications in Medical Care. New York, NY: IEEE Press.

Minsky, M. 1975. A Framework for Representing Knowledge. In *The Psychology of Computer Vision*, ed. P. Winston. New York, NY: McGraw-Hill.

Nielson and Shriver, eds. 1990. *Visualization in Scientific Computing.* Los Alamitos, NM: IEEE Press.

Sacerdoti, E. 1975. The nonlinear nature of plans. In *Advance Papers from the Fourth Int. Joint Conference on Artificial Intelligence*, Los Altos, CA: Morgan Kaufman.

Schank, R. and Abelson, R. 1977. *Scripts, Plans, Goals and Understanding.* Hillsdale, NJ: Lawrence Erlbaum.

Schank, R. 1982. *Dynamic Memory: A theory of learning in computers and people.* New York, NY: Cambridge University Press.

Schank, R. and Riesbeck, C. 1989. *Inside Case-Based Reasoning.* Hillsdale, NJ: Lawrence Erlbaum.

Searles, J. 1993. Investigating the linguistics of DNA with Definite Clause Grammars. In *Logic Programming: Proceedings of the North American Conference on Logic Programming,* ed. E. Lusk and R. Overbeek. Cambridge, MA: MIT Press.

13.

Appendix A - ISCN Grammars

13.1 The ISCN Short Form Grammar

This is the listing of the context free grammar for ISCN short form expressions, expressed as productions in T-gen format.

```
ChSet :
        ChCnt                                  {ChSet}
        | ChCnt ',' SexList                    {ChSet}
        | ChCnt ',' AbnormList                 {ChSet}
        | ChCnt ',' SexList ',' AbnormList  {ChSet}
        ;

 ChCnt :
        Num                                    {ChCnt};

SexList :
        XList YList                            {SexList}
        | XList                                {SexList}
        | YList                                {SexList}
        ;

XList :
        ChX XList                              {liftRightChild}
        | ChX                            {XList}
        ;

YList :
        ChY YList                        {liftRightChild}
        | ChY                            {YList}
        ;

AbnormList :
        Abnorm ',' AbnormList            {liftRightChild}
        | Abnorm                         {AbnormList}
        ;

Abnorm :
        NumericAbnorm                    {NumericAbnorm}
        | StructuralAbnorm               {StructuralAbnorm}
        ;

" NUMERIC ABNORMS "

NumericAbnorm :
         '+'  '?' AnyChrom               {AddUncertainCh}
        | '+'  '?' AnyChrom Sign         {AddUncertainChOfDiffLength}
        | '+' AnyChrom                   {AddCh}
        | '+' AnyChrom Sign              {AddChOfDiffLength}
        | '-'  AnyChrom                  {MissCh}
        | '-'  '?' AnyChrom              {MissUncertainCh}
        | '+' ChPartOfDiffLength         {AddChPartOfDiffLength}
        | ChPartOfDiffLength
        ;

ChPartOfDiffLength :
        AnyChrom ChPartDownToRegion Sign {ChPartOfDiffLength}
        ;

" *************************************** "
" STRUCTURAL ABNORMS "

StructuralAbnorm :
        Translocation                    {Translocation}
        | Deletion                       {Deletion}
        ;

" TRANSLOCATIONS "

Translocation :
        't' TwoChromTwoBreakRea
        ;

" DELETION "
Deletion :
        'del' UpToTwoBreakRea
        ;
```

```
" REARRANGEMENTS "

UpToTwoBreakRea :
      '(' Autosome ')' '('
          ChPartUpToArm ')'                  {UpToTwoBreakRea}
      | '(' Autosome ')' '('
          ChPartUpToArm ChPartUpToArm ')'    {UpToTwoBreakRea}
      ;

TwoChromTwoBreakRea :
      '(' Autosome ';' Autosome ')' '(' ChPartUpToArm
          ';' ChPartUpToArm ')'              {TwoChTwoBreakRea}
      ;


 "*****************************************"
" CHROMOSOME PARTS"


ChPartDownToRegion :
        Arm                               {ChPartDownToRegion}
        | Region                          {ChPartDownToRegion}
        ;

ChPartUpToArm :
        Band                              {ChPartUpToArm}
        | Region                          {ChPartUpToArm}
        | Arm                             {ChPartUpToArm}
        ;

Band :
        Region '.' Num                    {Band};

Region :
        Arm Num                           {Region};

Arm :
        'p'                               {PArm}
        | 'q'                             {QArm}
        ;

AnyChrom :
        SexChrom
        | Autosome;

Autosome :
        Num                               {Autosome};

SexChrom :
        ChX                               {SexChrom}
        | ChY                             {SexChrom}
        ;

ChX :
        'x'                                 {ChX};

 ChY  :
        'y'                                 {ChY};

Sign  :
        '+'                                 {Plus}
        |  '-'                              {Minus}
        ;

Num  :
        <num>                               {Num};
```

Listing 13.1 ISCN Short Form Grammar

13.2 The ISCN Long Form Grammar

This is the listing of the context free grammar for ISCN long form expressions, expressed as productions in T-gen format.

```
ISCNLongExpr :
        BandSection '::' ISCNLongExpr       {liftRightChild}
        | BandSection                       {ISCNLongExpr}
        | BandEnd                           {ISNLongExpr}
        ;

BandSection :
        StartBand '->' EndBand              {BandSection};

StartBand :
        BandEnd                             {StartBand};

EndBand :
        BandEnd                             {EndBand};

BandEnd :
        Centromere
        | Terminal
        | Region
        | Band
        ;

 "*****************************************"
" CHROMOSOME PARTS"
```

```
Terminal :
        Ch 'pter'                        {PTerminal}
        | Ch 'qter'                      {QTerminal}
        ;

Band :
        Region '.' Number                {Band};

Region :
        Ch Arm Number                    {Region};

Arm :
        'p'                              {PArm}
        | 'q'                            {QArm}
        ;

Centromere :
        'cen'                            {Centromere};

Ch :
        Num                              {Ch}
        | 'x'                            {Ch}
        | 'y'                            {Ch}
        ;

Number :
        Num                              {Number};

Num :
        <num>                            {Num};
```

Listing 13.2 ISCN Long Form Grammar