A TEMPORAL FRAMEWORK FOR INFORMATION SYSTEMS
SPECIFICATION AND VERIFICATION

DAISEE; Working Paper No. 31

Kung Chen ho (Gong Zhen-he)

Arne Sølvberg

A formal framework for information systems specification and verification is proposed.

The framework for specification consists of three parts: the specification of the static and temporal constraints and the description of the operations.

The verification of the static constraints ensures that there will be at least one legal information system state in which all the static constraints are satisfied. The operation analysis checks for that each single operation can ever be executed in the future information system. The verification of the temporal constraints checks for that each sequence of operations that is allowed to be executed in the future information system will satisfy every temporal constraint.

Some notions and problems concerning information modelling and consistency checking are also discussed in the thesis.

Inform. Syst Specification
Formal Verification
Temporal Dimension
A TEMPORAL FRAMEWORK FOR
INFORMATION SYSTEMS SPECIFICATION AND VERIFICATION

by

KUNG CHEN HO *

Thesis submitted to the Department of Computer Science, The Norwegian Institute of Technology, in partial fulfillment of the requirements for the degree of Doctor of Engineering.

DEPARTMENT OF COMPUTER SCIENCE
THE NORWEGIAN INSTITUTE OF TECHNOLOGY
UNIVERSITY OF TRONDHEIM
THE KINGDOM OF NORWAY

* On leave from the Chinese Academy of Sciences, Beijing, China.
TO CHINA AND NORWAY
PREFACE

This work has been performed at the Department of Computer Science in the University of Trondheim, Norway. It fulfills the research work for the degree of Doctor of Engineering.

I wish to express my deep gratitude to my supervisor Prof. Arne Sølvberg for his continual guidance and encouragement. His suggestions have been of immense help in the course of this research. His patience in reading and improving the earlier drafts of this report is also greatly appreciated. Through the contact with Prof. Sølvberg, I have learned, not only what is reported in this thesis, but more about how to do research and the attitude to one's job. Time will pass and what has been done in this research could be forgotten. However, the time that I have been working under the guidance of Prof. Sølvberg will always remain in my memory.

I also wish to thank Prof. Chisung Tang, who is working in the Chinese Academy of Sciences, for his kindly concern and encouragement. His recommendation gave me the opportunity to study in Norway.

I have stayed geographically away from my home country for the last few years. During this period, immense improvement and progress have taken place in my motherland almost every minute. Economic reform and the application of science and technology is upgrading drastically the living standard in our ancient country. China's attitude towards advanced science and technology has never been so eager, serious and practical. I would like to say that the steady development of China's economy has continually encouraged me in carrying out my study here.

This research work has been financially supported by the Norwegian Agency for International Development, NORAD. Through the Department of Computer Science, the Norwegian people have provided me with the best research environment. In this respect, I would like to thank Dr. Axel Lassen, Berit Fantoft, Turid Bråk and Daphne Tangen for their assistance in solving my day to day problems. Through NORAD, I would like to express my deep gratitude to the Norwegian people. I hope that the friendship and corporation between the two
countries will be ever growing.
I also wish to thank the whole staff of the Department of Computer Science and the staff of the Computing Center of the University of Trondheim (RUNIT). They have been so kind in helping me in various aspects. Specifically, I would like to thank Prof. Kristen Rekdal for his arrangement of my study in Norway and his continuous concern during the last years. Special thanks also to the secretaries of the Department of Computer Science, Ms. Norun Bremsen, Miss Else Johanne Svorkis and Ms. Anne Johanne Særvik. In particular, they helped me by retyping the first two chapters which were destroyed by my carelessness.

Thanks also to Svein Stavelin, Jon Ola Hove and Tore Amble for constructive discussions.

Finally but not the least, I am deeply indebted to my wife Zhao Xinyu. She has been responsible for almost all the housework besides her own study. Without her patience and encouragement, I could never have achieved so much in this research period.

Trondheim, April 1984.

[Signature]

Kung Chen Ho
ABSTRACT

Few reports have been published on formal verification of information systems specifications, specifically when the temporal dimension of information modelling is taken into account. It is assumed that formal verification of information systems specifications may reduce the total systems cost.

A formal framework for information systems specification and verification is proposed.

The framework for specification consists of three parts: the specification of the static constraints, the specification of the temporal constraints and the specification of the information systems operations.

The framework for verification also consists of three parts: the verification of the static constraints, the analysis of the operation descriptions and the verification of the temporal constraints.

The verification of the static constraints ensures that there will be at least one legal information system state in which all the static constraints are satisfied. The operation analysis checks for that each single operation can ever be performed in the future information system. The verification of the temporal constraints checks for that each sequence of operations that is allowed to be performed in the information system will satisfy every temporal constraint.

Some notions and problems concerning information modelling and consistency checking are also discussed in the thesis.
# TABLE OF SYMBOLS

## FIRST ORDER LOGIC:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(, )</td>
<td>parentheses</td>
</tr>
<tr>
<td>u, v, x, y, z</td>
<td>variables</td>
</tr>
<tr>
<td>u, v, x, y, z</td>
<td>parameters</td>
</tr>
<tr>
<td>&amp;</td>
<td>logical and</td>
</tr>
<tr>
<td>V</td>
<td>logical or</td>
</tr>
<tr>
<td>→</td>
<td>logical implication</td>
</tr>
<tr>
<td>¬</td>
<td>negation</td>
</tr>
<tr>
<td>∀</td>
<td>universal quantifier</td>
</tr>
<tr>
<td>∃</td>
<td>existential quantifier</td>
</tr>
<tr>
<td>∃!</td>
<td>the existential quantifier under the finite domain assumption, §4.5.2</td>
</tr>
<tr>
<td>w</td>
<td>a well-formed formula or a temporal assertion</td>
</tr>
<tr>
<td>W</td>
<td>a set of well-formed formulae</td>
</tr>
<tr>
<td>L</td>
<td>the first order language, §3.1.1</td>
</tr>
<tr>
<td>L/W</td>
<td>the first order language whose non-logic symbols are those that appear in W, where W denotes a set of sentences, §3.1.4</td>
</tr>
<tr>
<td>=</td>
<td>identity of the first order language</td>
</tr>
<tr>
<td>s(c)</td>
<td>the sort of constant symbol c, §4.7.1</td>
</tr>
<tr>
<td>s(x)</td>
<td>the sort of variable x, §4.7.1</td>
</tr>
<tr>
<td>⊢</td>
<td>theorem, deducible (a syntactic notion), §3.1.4</td>
</tr>
<tr>
<td>α, β</td>
<td>literals, primes, i.e., atomic formulae or negated atomic formulae, see §5.1 or Appendix B</td>
</tr>
<tr>
<td>C_i</td>
<td>a clause which is a disjunction of literals</td>
</tr>
<tr>
<td>C</td>
<td>set of clauses</td>
</tr>
<tr>
<td>T</td>
<td>theory in a language which is a set of sentences, (§3.1.5)</td>
</tr>
<tr>
<td>Б</td>
<td>substitution, see §4.9.1, Appendix B</td>
</tr>
<tr>
<td>Φ</td>
<td>set of literals that should remain held in the new state after applying an operation, see §5.7.3</td>
</tr>
<tr>
<td>ψ</td>
<td>the same as for Φ except that update operations are concerned here, see §5.7.6</td>
</tr>
<tr>
<td>Π</td>
<td>logical consequences of a set of clauses, (§5.7.2)</td>
</tr>
<tr>
<td>Ψ</td>
<td>set of literals which are logical consequences of a set of clauses, §5.7.2</td>
</tr>
<tr>
<td>sc</td>
<td>a static constraint</td>
</tr>
<tr>
<td>SC</td>
<td>a set of static constraints</td>
</tr>
</tbody>
</table>

## MODEL THEORY:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>structure of a language, §3.1.3</td>
</tr>
<tr>
<td>C_i, S_j</td>
<td>information systems states</td>
</tr>
<tr>
<td>U</td>
<td>the universe of a structure which is a non-empty set of individuals, §3.1.3</td>
</tr>
<tr>
<td>I</td>
<td>the interpretation component of a structure, (§3.1.3)</td>
</tr>
<tr>
<td>PS(k)</td>
<td>the parameter space of sort k which is a set of</td>
</tr>
</tbody>
</table>
individuals of sort $k$ introduced so far, §4.7.1

t($x_1, \ldots, x_n$) - a term $t$ whose variables are among $x_1, \ldots, x_n$. 
§3.1.4

t($t_1, \ldots, t_n$) - the value of term $t$ when its variables take
as values $t_1, \ldots, t_n$, respectively, where
$t_1, \ldots, t_n$ are terms, §3.1.4

w($x_1, \ldots, x_n$) - a well-formed formula whose variables are among $x_1, \ldots, x_n$. §3.1.4

w($t_1, \ldots, t_n$) - the value of the well-formed formula $w$ when its
variables take as values $t_1, \ldots, t_n$, respectively, where$t_1, \ldots, t_n$ are terms, §3.1.4

$\forall$ - validity, consequence (a semantic notion), §3.1.4

$\mu$ - associate set, see §4.8.1

$\exists$ - alternative set, see §4.8.2, §4.8.3

$H$ - Herbrand universe, §5.3

SET THEORY:

$\{,\}$ - set parentheses

$\in$ - belongs to, set membership relation

$\notin$ - not belong to, set membership relation

$\subseteq$ - improper set inclusion

$\cap$ - set intersection

$\times^n$ - the $n$-ary Cartesian product of set $S$

TEMPORAL LOGIC:

always$^+$ - always in the past excluding the present, §3.2.3

always$^*$ - always in the past including the present, §3.2.3

always$^+$ - always in the future excluding the present, §3.2.3

always$^+$ - always in the future including the present, §3.2.3

ever$^+$ - ever in the past excluding the present, §3.2.3

ever$^+$ - ever in the past including the present, §3.2.3

ever$^+$ - ever in the future excluding the present, §3.2.3

ever$^+$ - ever in the future including the present, §3.2.3

$\mathsf{aux}$ - a dummy temporal assertion which is implied by
every temporal assertion, §3.2.3

w - a well-formed formula or a temporal assertion

tc - a temporal constraint

TC - a set of temporal constraints

EXECUTABLE - an abbreviation of three conditions for executing an
operation, see §3.2.3

$\Gamma_p$ - set of partial execution sequences, §4.10.5

$\Gamma$ - set of total execution sequences, §4.10.5

$\tau$ - set of test sequences, §4.10.3.2

$\sigma$ - a partial execution sequence, which is a
sequence of information system states, see §§3.2.3

$\sigma$ - a total execution sequence, which is a sequence of
information system states, see §§3.2.3 and 4.10.5

AUTOMATA THEORY:
$\Omega$ - set of operations which are the input symbols of the finite automata, see §4.10.1

$\delta$ - a mapping from one operation and one state to one state, see §§3.2.3 and 4.10.1

$\delta'$ - a mapping from one operation, one state and one pushdown symbol to a pair of one state and one pushdown symbol, §4.10.2.4

$\lambda$ - a mapping of a pair of temporal assertions to another temporal assertion, see §§4.10.2.2 and 4.10.2.3

$Z$ - a special pushdown symbol, which implies every temporal assertion, §4.10.2

$F$ - the final states of an automaton, §4.10.1

$K$ - the set of states of an automaton, §4.10.1

$\Gamma$ - the set of pushdown symbols of a pushdown automaton, the pushdown symbols are temporal assertions, §4.10.2

$fa_i$ - the finite automaton with state $S_i$ as the initial state, §4.10.1

$pa_i$ - the pushdown automaton with state $S_i$ as the initial state, §4.10.2

**METALANGUAGE SYMBOLS:**

`pre(<op>)` - the precondition of the operation `<op>`

`post(<op>)` - the postcondition of the operation `<op>`

`<=>` - metalanguage equivalence

`==` - metalanguage identity
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 INTRODUCTION</td>
<td></td>
</tr>
<tr>
<td>1.1 On the Title</td>
<td>1</td>
</tr>
<tr>
<td>1.2 On the Problem</td>
<td></td>
</tr>
<tr>
<td>1.2.1 The Three-realm Doctrine</td>
<td>2</td>
</tr>
<tr>
<td>1.2.2 The Role of a Conceptual Model</td>
<td>3</td>
</tr>
<tr>
<td>1.2.3 Desirable Features of a Conceptual Model</td>
<td>4</td>
</tr>
<tr>
<td>1.2.4 Definition of Information Systems Specification</td>
<td>5</td>
</tr>
<tr>
<td>1.3 The Aim and the Scope</td>
<td></td>
</tr>
<tr>
<td>1.3.1 Modelling Approach and Specification Language</td>
<td>6</td>
</tr>
<tr>
<td>1.3.2 The Aim and the Scope</td>
<td>7</td>
</tr>
<tr>
<td>1.4 The Layout of the Thesis</td>
<td>8</td>
</tr>
<tr>
<td>2 INFORMATION SYSTEMS SPECIFICATION AND VERIFICATION</td>
<td>10</td>
</tr>
<tr>
<td>2.1 Modelling Approaches Models Reality and Validity</td>
<td></td>
</tr>
<tr>
<td>2.1.1 A Note on &quot;Model&quot;</td>
<td>10</td>
</tr>
<tr>
<td>2.1.2 Model and Reality</td>
<td>11</td>
</tr>
<tr>
<td>2.1.3 From Reality to Model</td>
<td>11</td>
</tr>
<tr>
<td>2.1.4 From Model to Reality</td>
<td>13</td>
</tr>
<tr>
<td>2.1.5 Summary Remarks</td>
<td>14</td>
</tr>
<tr>
<td>2.2 On the Non-uniqueness of Conceptual Models</td>
<td>14</td>
</tr>
<tr>
<td>2.3 Using Mathematical Logic as a Specification Language</td>
<td>16</td>
</tr>
<tr>
<td>2.3.1 Advantages</td>
<td>17</td>
</tr>
<tr>
<td>2.3.2 Disadvantages</td>
<td>17</td>
</tr>
<tr>
<td>2.4 Modelling in the Temporal Dimension</td>
<td>18</td>
</tr>
<tr>
<td>2.4.1 Four Types of Approaches</td>
<td>18</td>
</tr>
<tr>
<td>2.4.2 Some Temporal Framework</td>
<td>19</td>
</tr>
<tr>
<td>2.5 On Consistency Checking of Specifications</td>
<td>21</td>
</tr>
<tr>
<td>2.5.1 Informal Discussion of Verifying Static Constraints</td>
<td>21</td>
</tr>
<tr>
<td>2.5.2 Analysis of Operation Descriptions</td>
<td>22</td>
</tr>
<tr>
<td>2.5.3 Consistency Checking of Temporal Constraints</td>
<td>22</td>
</tr>
<tr>
<td>2.6 Existing Works in Consistency Checking</td>
<td>25</td>
</tr>
<tr>
<td>2.6.1 Consistency Checking in Information System Field</td>
<td>25</td>
</tr>
<tr>
<td>2.6.2 Consistency Checking in the Database Field</td>
<td>27</td>
</tr>
<tr>
<td>2.6.3 Semantic Constraints and Integrity Constraints</td>
<td>28</td>
</tr>
<tr>
<td>2.6.4 Consistency Checking and Integrity Enforcement</td>
<td>29</td>
</tr>
</tbody>
</table>
### 3 THE FRAMEWORK FOR SPECIFYING INFORMATION SYSTEMS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Static Constraints in a First Order Language</td>
<td>33</td>
</tr>
<tr>
<td>3.1.1 The First Order Language</td>
<td>33</td>
</tr>
<tr>
<td>3.1.2 The First Order Axioms</td>
<td>35</td>
</tr>
<tr>
<td>3.1.3 The Concepts of Structure and Model</td>
<td>36</td>
</tr>
<tr>
<td>3.1.4 The Notion of Satisfaction</td>
<td>36</td>
</tr>
<tr>
<td>3.1.5 First Order Theory and Conceptual Model</td>
<td>38</td>
</tr>
<tr>
<td>3.1.6 A Simple Example</td>
<td>39</td>
</tr>
<tr>
<td>3.2 The Specification of Temporal Constraints</td>
<td>40</td>
</tr>
<tr>
<td>3.2.1 The Temporal Language</td>
<td>40</td>
</tr>
<tr>
<td>3.2.2 On Parameters</td>
<td>42</td>
</tr>
<tr>
<td>3.2.3 The Semantics</td>
<td>42</td>
</tr>
<tr>
<td>3.2.4 Examples</td>
<td>44</td>
</tr>
<tr>
<td>3.3 Operation Description</td>
<td>44</td>
</tr>
<tr>
<td>3.3.1 The Formal Definition of an Operation</td>
<td>45</td>
</tr>
<tr>
<td>3.3.2 Some Examples</td>
<td>45</td>
</tr>
<tr>
<td>3.3.3 The Formal Semantics</td>
<td>46</td>
</tr>
<tr>
<td>3.3.4 Related Work in Artificial Intelligence</td>
<td>47</td>
</tr>
<tr>
<td>4.1 The Monkey and Bananas Example</td>
<td>48</td>
</tr>
<tr>
<td>4.2 Comparative Remarks</td>
<td>50</td>
</tr>
</tbody>
</table>

### 4 CONSISTENCY CHECKING: THE MODIFIED TABLEAUX APPROACH

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Syntactical and Semantical Approaches</td>
<td>53</td>
</tr>
<tr>
<td>4.2 An Overview of the Modified Tableaux Approach</td>
<td>54</td>
</tr>
<tr>
<td>4.2.1 Verification of Static Constraints</td>
<td>54</td>
</tr>
<tr>
<td>4.2.2 Operation Analysis</td>
<td>54</td>
</tr>
<tr>
<td>4.2.3 Verification of Temporal Constraints</td>
<td>55</td>
</tr>
<tr>
<td>4.2.4 The Layout of the Chapter</td>
<td>55</td>
</tr>
<tr>
<td>4.3 The Unsolvability of First Order Logic</td>
<td>55</td>
</tr>
<tr>
<td>4.4 The First Order Tableaux System</td>
<td>56</td>
</tr>
<tr>
<td>4.4.1 The Deduction Rules</td>
<td>56</td>
</tr>
<tr>
<td>4.4.2 The Tableaux Method</td>
<td>59</td>
</tr>
<tr>
<td>4.4.3 An Example</td>
<td>59</td>
</tr>
<tr>
<td>4.4.4 Summary Remarks</td>
<td>61</td>
</tr>
<tr>
<td>4.5 Restricting the Static Constraints</td>
<td>62</td>
</tr>
<tr>
<td>4.5.1 Using Many-sorted Logic</td>
<td>62</td>
</tr>
<tr>
<td>4.5.2 Excluding Recursiveness</td>
<td>64</td>
</tr>
<tr>
<td>4.6 Detection of Recursiveness</td>
<td>65</td>
</tr>
<tr>
<td>4.6.1 Prenex Form</td>
<td>65</td>
</tr>
<tr>
<td>4.6.2 The D-graph</td>
<td>66</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>4.7 The Modified Tableaux Approach</td>
<td>66</td>
</tr>
<tr>
<td>4.7.1 The Modified Deduction Rules</td>
<td>66</td>
</tr>
<tr>
<td>4.7.2 The Modified Method</td>
<td>67</td>
</tr>
<tr>
<td>4.7.3 Summary Discussions</td>
<td>70</td>
</tr>
<tr>
<td>4.8 Construction of System States</td>
<td>72</td>
</tr>
<tr>
<td>4.8.1 The Associate Sets</td>
<td>72</td>
</tr>
<tr>
<td>4.8.2 The Semantics of the Associate Sets</td>
<td>73</td>
</tr>
<tr>
<td>4.8.3 Formal Definition of Alternative Sets</td>
<td>74</td>
</tr>
<tr>
<td>4.8.4 An Example</td>
<td>75</td>
</tr>
<tr>
<td>4.8.5 Construction of Legal System States</td>
<td>75</td>
</tr>
<tr>
<td>4.9 Analysis of Operations</td>
<td>76</td>
</tr>
<tr>
<td>4.9.1 A Short Note on Unification</td>
<td>75</td>
</tr>
<tr>
<td>4.9.2 Informal Introduction to Operation Analysis</td>
<td>76</td>
</tr>
<tr>
<td>4.9.3 Formal Analysis of Insert and Delete Operations</td>
<td>77</td>
</tr>
<tr>
<td>4.9.4 The Description of an Example</td>
<td>78</td>
</tr>
<tr>
<td>4.9.5 Example: Analysis of an Insert Operation</td>
<td>79</td>
</tr>
<tr>
<td>4.9.6 Example: Analysis of a Delete Operation</td>
<td>81</td>
</tr>
<tr>
<td>4.9.7 Analysis of Update Operations</td>
<td>82</td>
</tr>
<tr>
<td>7.1 Informal Introduction</td>
<td>83</td>
</tr>
<tr>
<td>7.2 Formal Analysis of Update Operations</td>
<td>84</td>
</tr>
<tr>
<td>7.3 Example: Analysis of an Update Operation</td>
<td>85</td>
</tr>
<tr>
<td>4.9.8 State Transition Diagram</td>
<td>85</td>
</tr>
<tr>
<td>4.9.9 Interpretation of The Transition Diagram</td>
<td>86</td>
</tr>
<tr>
<td>4.10 Verification of Temporal Constraints</td>
<td>87</td>
</tr>
<tr>
<td>4.10.1 The Finite Automata</td>
<td>87</td>
</tr>
<tr>
<td>4.10.2 The Pushdown Automata</td>
<td>88</td>
</tr>
<tr>
<td>2.1 Some Temporal Axioms</td>
<td>88</td>
</tr>
<tr>
<td>2.2 Informal Presentation of the Lambda Mapping</td>
<td>89</td>
</tr>
<tr>
<td>2.3 Formal Definition of the Lambda Mapping</td>
<td>90</td>
</tr>
<tr>
<td>2.4 The Delta' Mapping</td>
<td>90</td>
</tr>
<tr>
<td>2.5 The Behavior of the Pushdown Automata</td>
<td>90</td>
</tr>
<tr>
<td>4.10.3 Generate Test Sequences</td>
<td>91</td>
</tr>
<tr>
<td>3.1 The Testing Tree</td>
<td>91</td>
</tr>
<tr>
<td>3.2 Test Sequences</td>
<td>92</td>
</tr>
<tr>
<td>4.10.4 Acceptance of Test Sequences</td>
<td>92</td>
</tr>
<tr>
<td>4.1 Example: the Rejection of a Test Sequence</td>
<td>92</td>
</tr>
<tr>
<td>4.2 Example: The Acceptance of a Test Sequence</td>
<td>93</td>
</tr>
<tr>
<td>4.10.5 Execution Sequences</td>
<td>93</td>
</tr>
<tr>
<td>4.10.6 Verification of the Temporal Constraints</td>
<td>94</td>
</tr>
<tr>
<td>4.11 Summary Discussions</td>
<td>95</td>
</tr>
</tbody>
</table>

5 CONSISTENCY CHECKING: AN IMPROVED APPROACH

5.1 Lewis' Work
Section | Page
--- | ---
5.2 Our Definition of Compactness
  5.2.1 Terminology for Directed Graphs | 100
  5.2.2 Unifiability Digraph | 100
  5.2.3 Proof of a Theorem | 101

5.3 Some Results for G(S) | 103

5.4 Example 1 | 105

5.5 Example 2 | 106

5.6 Example 3 | 107

5.7 Operation Analysis in the Improved Method
  5.7.1 Informal Discussion of Basic Ideas | 111
  5.7.2 Proofs of Two Theorems | 114
  5.7.3 Formal Analysis of Insert and Delete Operations | 115
  5.7.4 The Example | 117
  5.7.5 Analysis of an Insert Operation | 117
  5.7.6 Formal Analysis of Update Operations | 119
  5.7.7 Example: The Analysis of an Update Operation | 120

5.8 Summary | 121

6 CONCLUSION AND FUTURE RESEARCH DIRECTIONS | 124

6.1 Claimed Contributions | 124

6.2 Limitations and Future Research Directions | 125

REFERENCES | 126

APPENDIX A: TRANSFORMATION TO CLAUSE FORM | 134

APPENDIX B: UNIFICATION AND UNIFICATION PROCESS | 137

APPENDIX C: AN IMPROVED PROOF METHOD | 140
CHAPTER 1:

INTRODUCTION
1 INTRODUCTION

1.1 On the Title

As the title suggests, the aim of this thesis is to propose a formal framework for testing the consistency of information system specifications. There are various interpretations of what is meant by the phrase "testing the consistency of information system specifications", some of which are quite different from the one that is the startpoint of this thesis. This situation is due to the following reasons:

- The word "specification" has been used with two distinct meanings in the computer literature. The first is the engineering usage: a specification is a precise statement of the requirements that a product must satisfy. The second is defined in the dictionary which has a broader meaning that includes in a specification any additional information about the object being specified to make the description of the object more specific [103]. In the information system field, the former usage of the word specification results in the specification of information and performance requirements only. This type of specification does not include a conceptual description of the object system for which the information system is built. Until late 1970's, the research community could only gradually agree on the inclusion of a conceptual description of the object system [18] [20] [10] [88] [102] [89] [96] [99]. This usage of the word corresponds to the second meaning of "specification". Clearly, this historic development has significant impact on the interpretation of the concept of consistency of an information system specification.

- In addition to the above distinction, the phrase, "information system specification" has, sometimes during the 1970's been defined in terms of an executional semantics. This means that the conceptual model of the information system has focused on the processes, the flow of control and of data. The PSL/PSA technique [97], the Software Requirement Engineering Methodology (SREM) [8] [29], the Structured Analysis and Design (SAD) [81] [37] all belong to this category. This kind of approach has been called "solution-oriented" approach by Bubenko [19] [20] [21]. It is called "procedural specification" by some other authors [85] [84] [57] [53]. Opposite to the procedural approach, a tendency toward "declarative" approaches has appeared in a number of fields [102] [20] [84] [103]. These different strategies of specification have a significant impact on the understanding of the consistency checking.

- The word "consistency" has been defined and used by many different people with quite different meanings. Some of the communities of people which have defined and used the word "consistency" in their respective fields are:

  - The mathematical community (e.g., [9] [23] [33]).
- The database community (e.g., [10] [30] [44] [87] [11]).
- The software design and specification community (e.g., [103]).
- The artificial intelligence community (e.g., [72] [50] [36]).
- The information modelling community (e.g., [18] [64] [5] [58] [51] [52]).

Each of these groups assigns similar or completely different meanings to the word "consistency". Moreover, some researchers have defined "consistency" in a specific way (e.g. [73] [74]). These differences in the meaning of the word "consistency" result in different understandings of what is meant by consistency checking.

Despite the development in data modelling and information modelling, the qualitative aspects such as model validity, consistency and reliability has yet received very little attention from the research community. Although this problem has already been observed by Bubenko in 1977 [18], the progress has been relatively slow. Few results have been published on formal verification of information systems specifications, specifically when the temporal dimension is taken into account. This situation makes consistency checking less familiar to the information modelling scientists.

1.2 On the Problem

1.2.1 The Three-realm Doctrine

The above discussion indicates that prior to the presentation of the consistency checking method, we must agree on what is meant by

- "information system specification"; and
- "the consistency of an information system specification".

As pointed out in [17] [18] [88] and also from a database design point of view in [99] [3], the design of an information system has to consider three realms:

- The real world realm which includes the object system as a part that is relevant to the application.
- The conceptual realm which includes a so-called conceptual model of the relevant part of the application. (It is called "conceptual information model" by Bubenko and his colleagues to distinguish it from "conceptual database model".)
- The datalogical realm which corresponds roughly to the storage structure of the data in a data base.

In [24], a 4-level view is presented. The first 2 levels of [24] correspond roughly to the conceptual realm above. The last 2 levels may be seen as the datalogical realm.
In agreement with [83], we distinguish the notions "data model" and "conceptual model" in the following way. A model is called a conceptual model if it is suitable for representing somebody's perception of some slice of reality. This conceptual model consists of both structure and operation. We call a model a data model if it takes primarily into account in which way data are arranged and (efficiently) manipulated. It does not necessarily represent somebody's perception of a slice of reality.

1.2.2 The Role of a Conceptual Model

The conceptual model in the conceptual realm is an abstract representation of the real world phenomena that are of interest to the application in question. The conceptual model is usually constructed by using a modelling approach during the system analysis phase. The role of such a conceptual model has been addressed by many researchers [17] [20] [89] [51] [64] [10] [108] [85]:

- The conceptual model serves as a model of reality, which gives insight into the application domain. In other words, the construction of the conceptual model enables the system analysts to have a better understanding of the application and the users needs. As shown in [26] and pointed out in [20], faults that are introduced during the early system development stages (system analysis and design) have a major impact on the system development effort. It is shown in [7] that 64% of the total system development errors are analysis and design errors, in contrast to 36% of coding errors. That is, errors introduced in the analysis phase have the greatest impact if their discovery occurs in the maintenance phase. Unfortunately, it is shown in [28], that 45% of the errors that are found during or after acceptance test are analysis and design errors, in contrast to only 9% of coding errors. Thus a better understanding of the application and users needs by using a conceptual model seems to reduce the overall development cost.

- The conceptual model serves as a common reference framework which is used during the system analysis phase to communicate with the future users of the system. When specifying an information system, it is not uncommon that different local users will have different views of the "real world". As pointed out by Bubenko, "view integration often implies synthesis in the sense that new types can be created by generalization or by aggregation of other types. It may also imply decomposition. ...General principles for how to perform view integration are however still lacking and this task is therefore to a large extent still human art-work [20]." In [89], it was shown through an example that to a certain extent, view integration can be done in a conceptual model. In [86] and [87], the real world is conceptualized by aggregation and generalization which, among other benefits, allow the effective integration of different views.

- The conceptual model serves as a basis upon which the design and implementation of the database can be carried out and, against which the design and implementation can be tested.
- The conceptual model provides documentation of the system which can be used during the maintenance phase to facilitate modifications and enhancements of the system.

1.2.3 Desirable Features of a Conceptual Model

The role as defined above suggests the following features which should be supported by a conceptual model [51]:

- Understandability which includes aspects of:
  - user-oriented concepts and constructs;
  - readability of formal as well as informal descriptions,
  - unambiguity;
  - clarity; and
  - intuitivity.

- Expressiveness
  - power of expression;
  - resolution of detail; and
  - time perspective.

- Processing independence:
  - declarative specifications; and
  - descriptive reference of data.

- Checkability, which concerns
  - validity;
  - consistency; and
  - testability.

- Changeability:
  - the localization property; and
  - the loosely structured property.

Detailed definitions of these features can be found in [51], where this "feature framework" was used to access three conceptual models with time perspective. Some of the concepts in the above list will be
further discussed in the sequel.

1.2.4 Definition of Information Systems Specification

The above discussion leads to the definition of an information system specification as follows:

An information system specification consists of

- A conceptual model, in some formal language, of some part of reality that is of interest to the application in question.

- A requirement specification which includes
  - a statement in terms of the conceptual model, about the future information needs and
  - a statement about the performance criteria of the information system.

Depending on the modelling approach used, the conceptual models which might be eventually constructed might be quite different with respect to the following (incomplete) list of aspects:

- The level of abstraction can be different. E.g., the substantive level [91], the understanding level [20], the process/message level [84].

- The concepts and constructs can be different.

- The scope and the expressive power of the model can be different. E.g., pure static models [89] [24]; dynamic aspect is included [80] [4] [91] [66] [34] [10]; temporal dimension is included [22] [84] [85] [39]; and full time perspective is assumed [63]; [20].

- The way in which consistency checking is performed is different. E.g., some conceptual models allow a formal checking method while others only allow an informal way of consistency checking (e.g., experiments, prototyping).

Despite all these differences, the meaning of consistency checking of an information system specification should be the same. That is, an information system specification is consistent if

- the conceptual model description does not contain any conflicting statements; and

- the information needs in the requirement specification can be formally derived from the conceptual model. This aspect has been called "information derivability analysis" by Olive [73] [92] [74].
1.3 The Aim and the Scope

In this thesis, we propose a framework for specifying conceptual models which takes into account the temporal dimension of the object system. A formal method for testing the consistency of information systems specifications is also included.

1.3.1 Modelling Approach and Specification Language

It should be pointed out that the framework that is proposed here is only meant to be a formal specification language rather than a modelling approach. Although the border line between specification languages and modelling approaches usually cannot be clearly drawn, the difference between the two does exist.

A modelling approach concentrates on providing concepts and constructs for modelling (something). The correspondence between an instance of a construct of a modelling approach and an instance of the modelled object is stressed in a modelling approach. On the other hand, a specification language concentrates on the rules or grammar for forming the sentences of the language. The meaning of the sentences is obtained by interpreting the symbols of the language. The interpretation is usually specified informally.

The border line between a modelling approach and a specification language is vague because it is possible to use the language ingredients to model things and, it is also possible to use the modelling concepts and constructs to specify things.

In the information systems area, first order language is used as a modelling approach by some researchers. Examples of this kind are discussed in [46] and [83], where the advantages and disadvantages of such an approach are addressed. On the other hand, an instance of a (conceptual) data model or process model is very often used as a specification of an enterprise. This implies the use of a modelling approach as a specification language. Analysis of such approaches can be found in [20] [83] [51].

Examples which distinguish the modelling component and the specification component are found in [15] [84] [85] and [80].

In [15], a data type model is proposed for modelling the data in a database while a database schema specification language Beta is used to specify the integrity constraints.

In [85], a process/message model is proposed for modelling information systems, while a specification language INFOLOG is used to specify the concurrent processes. An information system is modelled by a network of concurrent processes that exchange messages among themselves and with the environment, through the temporal database of the information system. The specification language INFOLOG is designed to have a sound and complete mathematical foundation, i.e., it is based upon a multityped temporal logic.

In [80], an object-event-operation model is proposed for modelling applications while a specification language is proposed for specifying the events and operations.
The separation of modelling approaches from specification languages is a compromise made in the design process. According to the principles of the second generation systems approach [77], the conceptual model of an application for which the system is built plays the role of facilitating the mutual communication and understanding among the various parties that are involved in the development of the system. Thus, user-oriented modelling concepts and constructs are needed in a modelling approach [51]. The compromise that is made in this thesis is to use the framework merely as a specification language which must be supported by a modelling approach.

1.3.2 The Aim and the Scope

In this thesis, the specification of a conceptual model consists of three parts:

1) A set of first order logic sentences which together specifies a **snapshot** of the enterprise. This set of sentences must be satisfied by every **legal system state**.

2) A set of **temporal assertions** which specify the **temporal constraints** of the future information system. The temporal constraints have the property that every **sequence of operations** that is allowed to be performed (on the future information system) should satisfy all the temporal constraints.

3) A set of **operation descriptions** which specify the **preconditions** and the **postconditions** of the operations. Moreover, some temporal assertions are specified for the operations. By convention, a single operation is applicable if the precondition is satisfied by some system state. A sequence of operations is applicable only if the history of the information system satisfies the temporal assertion of each operation in the sequence at the time when the operation is to be applied.

The consistency checking framework also consists of three parts:

1) Consistency checking of the **static constraints**. By restricting the static constraints to a subset of well-formed formulae, the checking procedure can always determine the consistency or **inconsistency** of the static constraints.

2) Next, we analyse the operation descriptions to see if each single operation can possibly be executed in the future information system. That is, we examine if there is a legal system state in which the precondition of the operation holds and there is a legal system state in which the postcondition of the operation holds. These states are the **pre-state** and the **post-state** of the operation (conf. [43] [4] [50]). This analysis will result in a **state transition diagram** whose states are legal system states.

3) Finally, we test the consistency of the temporal constraints. That is, we check if each of the temporal constraints is satisfied by each sequence of operations that can ever be executed in the future information system. This is done in four steps:

i) First we transform the state transition diagram into a family of **finite automata** \( f_a_i \), \( i = 1, \ldots, n \), where \( n \) is the number of states of the transition diagram. For each \( f_a_i \), \( i = 1, \ldots, \ldots \).
n, we construct a pushdown automaton $p_{a_1}$ by taking into account the temporal assertions of the operations.

ii) Secondly we generate a set $T_1$ of test sequences for each $p_{a_i}$, $i = 1, \ldots, n$. A test sequence is a sequence of operations each of which (except the first one) can be performed in the state resulting from the application of the preceding operation, without considering the history of the system.

iii) Thirdly we use the pushdown automaton $p_{a_1}$ to analyse the acceptance of the test sequences in $T_1$. Intuitively, a sequence of operations can be executed in the future information system iff it is accepted by the pushdown automaton $p_{a_1}$. According to the theory of finite automata, a sequence of input symbols corresponds to a sequence of states of the finite automaton. Thus, for each test sequence that is accepted by some pushdown automaton $p_{a_i}$, there is a sequence of information system states, called a partial execution sequence because such an execution sequence records either the future behavior or the past behavior of the information system but not both. However, it is an easy task to form the total execution sequences from the set of partial execution sequences as we will see in Sect. 4.10.5.

iv) Finally, the consistency of the temporal constraints are verified iff each total execution sequence satisfies all the temporal constraints.

The information derivability analysis, however, will not be considered in this thesis.

1.4 The Layout of the Thesis

Chapter 2 discusses some of the issues concerning information modelling and consistency checking of information system specifications.

Chapter 3 presents the framework for specifying the conceptual model of the information system.

Chapter 4 describes the consistency checking method which is based on a modified tableaux approach. This method is straight-forward and hopefully easier to be understood.

Chapter 5 presents an improved approach for testing the consistency of the information systems specification. The approach is based on the works of Lewis [59]. It is an improvement upon the modified tableaux approach in the sense that it imposes the weakest restriction on the static constraints.

Chapter 6 concludes the study and points out some future research directions.
CHAPTER 2:

INFORMATION SYSTEMS

SPECIFICATION AND VERIFICATION
2 INFORMATION SYSTEMS SPECIFICATION AND VERIFICATION

Information modelling in the context of system development has been studied by a number of researchers [20] [83] [55] [46] [88] [89] [80] [84] [85] [13] [4] [51] [15]. Each of these researchers addresses a number of aspects of information modelling. However, a fundamental theory of information modelling has not yet been established. It is beyond the scope of the thesis to address all the fundamental problems of information modelling.

In this chapter, we will try to clarify some of the concepts and notions that are relevant to the field of consistency checking. We will also briefly review some of the recent developments in the field of information modelling and consistency checking.

2.1 Modelling Approaches Models Reality and Validity

2.1.1 A Note on "Model"

In the literature, the word "model" has been used with three distinct meanings. First, it is used to mean a modelling approach, i.e. a framework which can be used to describe a class of applications. For example, the relational model and the entity relationship model are to be understood in this sense.

The second use of the word "model" is to mean a product which represents a conceptual image of a piece of reality. This product is usually called a conceptual model, which is constructed by using a modelling approach and specified in some specification language.

The third meaning of the word is defined by mathematicians and is often found in connection with consistency checking [64] [50] [51] [52] [36]. Simply speaking, a model of a set of well-formed formulae in the mathematical sense consists of a universe and an interpretation. The universe is a non-empty set of individuals. The interpretation assigns an individual of the universe to each constant symbol, an operation to a function symbol and a relation to a predicate symbol such that all the formulae in the set are true under the interpretation.

An important result from model theory states that

"A set of well-formed formulae is consistent iff it has a model"

This result is commonly referred to as the Extended Completeness Theorem [23]. It will be used in Sect. 2.5.1. Its proof can be found in [23] [9].

Only the second meaning of the word "model" can be found in a dictionary. In this thesis, we will be as careful as possible in using this word. We will very often use the phrase "modelling approach" or "framework" instead of "model" when the first meaning of the word is intended. We will use "conceptual model" to denote the second meaning, and "legal system state" to denote the third meaning. When no
confusion can arise, we will use "system states" instead of "legal system states".

2.1.2 Model and Reality

As mentioned above, a conceptual model is usually specified in some specification language. A specification language is a collection of symbols called the alphabet of the language and a set of rules called the grammar of the language. The sentences of the language are constructed by using the symbols of the alphabet and the grammar of the language. A set of sentences of a language is called a theory in the language. A conceptual model is a set of sentences (in some specification language) which express someone's perception of a piece of reality. Therefore, a conceptual model can be regarded as a theory in the specification language which is used to describe the conceptual model [18] [20] [21] [38].

We are now at the position to discuss two of the fundamental problems concerning information modelling:

i) Given a piece of reality, how can one use a modelling approach to construct a conceptual model for it.

ii) Given a specification of some conceptual model in some specification language, how can one comprehend it.

The solution to the first problem is to define a mapping from the real world realm to the conceptual realm. That is, for each real world phenomenon, we define an instance of a construct of a specific modelling approach, this instance is supposed to be the representation of the real world phenomenon.

The solution to the second problem is to define a mapping from the conceptual realm to the real world realm. That is, to each instance of the constructs of a specification language, we associate with it a real world phenomenon. This association is commonly referred to be the interpretation of the specification of the conceptual model.

2.1.3 From Reality to Model

Suppose that we want to formally specify a conceptual model for some part of reality. This part of reality contains real world entities such as employees, managers, salaries, and some relationships between employees and managers, employees and salaries. One relationship between the employees and the managers is that employees work for managers. One relationship between the employees and the salaries is that employees have salaries.

Now how can we specify a conceptual model for such a part of reality in a formal language such as first order logic? One way of doing it is to define a mapping between the interesting part of reality and the formal language. Specifically

- Each real world entity $e_i$ is mapped to a constant symbol $c_i$ of the formal language. A class of real world entities $E_j$ is mapped to a unary predicate symbol $P_j$ of the language, with the convention that $P_j(c_i)$ is true if $e_i$ is a member of $E_j$ for the time being.
- Each real world relationship \( R_k \) between \( n \) real world entities \( e_1, \ldots, e_n \), is mapped to an \( n \)-ary predicate symbol \( P_k \) of the language with the convention that \( P_k(c_1, \ldots, c_n) \) is true if \( e_1, \ldots, e_n \) satisfy the relationship \( R_k \) for the time being, where \( e_1, \ldots, e_n \) are assumed to be mapped to \( c_1, \ldots, c_n \) respectively.

For example, the piece of reality above can be mapped to a formal language as follows:

- The entity class of employees is mapped to the unary predicate symbol \( \text{EMP} \). If an individual \( e_i \) is an employee for the time being and \( e_i \) is mapped to a constant symbol \( c_i \), then \( \text{EMP}(c_i) \) is true for the time being.

- The entity class of managers is mapped to the unary predicate symbol \( \text{MNG} \). If an individual \( e_j \) is a manager for the time being and \( e_j \) is mapped to a constant symbol \( c_j \), then \( \text{MNG}(c_j) \) is true for the time being.

- The relationship that employees work for managers is mapped to the binary predicate symbol \( \text{WF} \). If employee \( e_i \) works for manager \( m_j \) for the time being and \( e_i \) and \( m_j \) are mapped to the constant symbols \( c_i \) and \( c_j \) respectively, then \( \text{WF}(c_i, c_j) \) is true for the time being.

- The real world relationship that employees have salaries is mapped to the binary predicate symbol \( \text{HS} \). If employee \( e_i \) has salary \( s_j \) for the time being and \( e_i \) and \( s_j \) are mapped to the constant symbols \( c_i \) and \( c_j \) respectively, then \( \text{HS}(c_i, c_j) \) is true for the time being.

Since the perceived part of reality contains no particular individuals so far, our formal language at this stage has only four predicate symbols: \( \text{EMP}, \text{MNG}, \text{WF}, \text{and} \text{HS} \).

Suppose now that the perceived part of reality is extended to include two additional statements:

- Every manager is an employee.

- Every employee works for some manager.

Unlike the previous part of reality, which dealt only with concrete knowledge, this time the perceived part of reality includes some "abstract knowledge" [20] [64] [36]. This "abstract knowledge" reflects some general laws about reality [68]. It constrains the legal space of concrete knowledge for the time being. For example, with these two statements, any individual who is a manager must be an employee at the same time; and whenever an individual who is an employee must work for some manager. As a result, the conceptual model for this extended perception of reality must be extended in order to reflect these general laws.

One way of extending the existing conceptual model is to include the following well-formed formulae (wffs) into the conceptual model description:
\[(\forall x)(\text{MNG}(x) \rightarrow \text{EMP}(x))\]
\[(\forall x)(\text{EMP}(x) \rightarrow (\exists y)(\text{MNG}(y) \& \text{WF}(x,y)))\]

The above example shows that it is possible to define a mapping between a piece of reality and a formal language, such that the formal language can be used to specify a conceptual model for the chosen part of reality. However, the number of ways in which a conceptual model can be specified for a piece of reality in a formal language is in general, unlimited.

### 2.1.4 From Model to Reality

Another problem is that there are many possible interpretations of logical formulae. Suppose that we have a conceptual model description, in the first order language:

\[(\forall x)(\text{N}(x) \rightarrow (\exists y)(\text{P}(y) \& \text{G}(y,x)))\]

We do not know what this sentence tells us unless we know the "meaning" of the predicate symbols that are used in the description. Different meanings could have been intended for the predicate symbols:

1) \(\text{N}(x)\): \(x\) is a natural number.

\(\text{P}(x)\): \(x\) is a prime number.

\(\text{G}(y,x)\): \(y\) is greater than or equal to \(x\).

The interpretation of the sentence is that for every natural number \(x\) there exists a prime number \(y\) such that \(y\) is greater than or equal to \(x\).

2) \(\text{N}(x)\): \(x\) is an employee.

\(\text{P}(x)\): \(x\) is a project.

\(\text{G}(y,x)\): \(x\) works on \(y\).

The interpretation of the sentence is that every employee works on some project.

3) \(\text{N}(x)\): \(x\) is a man.

\(\text{P}(x)\): \(x\) is a woman.

\(\text{G}(y,x)\): \(y\) is the wife of \(x\).

The interpretation of the sentence is that every man has some woman as his wife.

4) \(\text{N}(x)\): \(x\) is a man.

\(\text{P}(x)\): \(x\) is a bank.

\(\text{G}(y,x)\): \(y\) is the father of \(x\).

The interpretation of the sentence is that every man has a bank as his father.
5) etc..

Clearly there is an infinite number of interpretations of the predicate symbols. However, not every interpretation results in a true statement. For example, in the first case above, the sentence tells us a theorem in mathematics, which has been proved to be true. In the third case however, the sentence does not tell us the truth because it is not the case that every man has a wife in general. In the last case, the sentence tells us something absurd according to the commonsense, and it is not true of course. The second case depends on the situation in which the statement is issued. That is, whether it is true or not depends on whether it is the case that every employee works on some project. This argument could have been applied to the third case. That is, if we issue the statement in the context of a marriage club, then the statement could have been true if it is the case that every man in a marriage club does have a woman as his wife.

2.1.5 Summary Remarks

The above discussion illustrates that a formal language can only enable one to express something by using the language. A sentence of a language does not necessarily express a truth, it may also happen to be an absurd sentence. Secondly, by reading a sentence expressed in a language, one does not know what the sentence is going to tell unless one knows the meaning of the "predicate symbols". We can communicate among ourselves in a language if the meanings of the "predicate symbols" of the language has been specified. Unfortunately, this can never be formally done as we cannot control the human cognitive process [10]. We can only agree on the meanings that are assigned to the predicate symbols. This agreement can only be made informally. As a consequence, any effort attempting to formally specify the relationship between a formal language and reality is hopeless, at least at the present. What we can do is to incorporate into the conceptual model certain amount of "knowledge" which is specified explicitly so that the comprehension of the specification can be as precise as desirable.

Research in data modelling and information modelling has mainly focused on finding better solutions to the above two problems. That is, to provide more suitable modelling constructs to better modelling the application and comprehend the conceptual model description.

The problem concerning whether a conceptual model correctly and adequately describes a piece of reality is defined as the validity of the conceptual model. The above discussion shows that the validity of a conceptual model can never be formally checked [18] [20]. The validity of a conceptual model can only be checked informally through negotiations, intuitions and experiments. If a conceptual model does not correctly and adequately describe a piece of reality, then the model specifier should modify the model specification until satisfactory result is obtained. This process is called the validation of a conceptual model which will not be addressed in this thesis.

2.2 On the Non-uniqueness of Conceptual Models

Given a piece of reality and a specification language, there are a number of conceptual models each of which is specified in the language and each of which correctly and adequately describes the piece of
reality. That is, the conceptual model of a piece of reality is not unique with respect to a specification language.

The non-uniqueness of conceptual model can be easily seen in a modelling approach using the first order language [83] [46]. In [83], an example concerning "the color of my shirt" is used to show the non-uniqueness of the conceptual models for this piece of reality. It is required to express the fact that "the color of my shirt is red". It is shown that at least two different conceptual models can be constructed within the framework of the first order predicate calculus.

The first conceptual model uses a unary predicate IS-RED(x) which is informally agreed to assert that the color of x is red. The conceptual model therefore appears as IS-RED(my-shirt).

The second conceptual model uses a binary predicate COLOR(x,y) which is informally agreed to assert that the color of x is y. That is, COLOR(my-shirt, red) will be the conceptual model of the piece of reality which we want to describe.

It is argued in [83] that one should avoid using such an unstructured language to describe a conceptual model. Schmid suggests that a modelling approach should directly incorporate some major constraints into the (language) constructs to get rid of a separately described list of constraints. That is, a more structured language is preferred in a modelling approach [83]. Schmid suggests that some major constraints should be incorporated directly into the conceptual model constructs for the following reasons:

1) We will know more accurately which interpretation is given to a thing that is represented by a construct.

2) The model designer should make explicit choices among a number of constructs.

3) Consistency constraints, which are listed otherwise in addition to the constructs at some places, will usually be expressed in some kind of predicate logic. However, the state of the art in "theorem-proving" does not allow at the moment, nor can it be expected in the near future, to derive and prove interesting results mechanically.

4) The different constraints that can be associated to the (more) general constructs must also be counted as primitives. Hence the number of total primitives are the same in both approaches.

We agree. However, we want to emphasize that:

1) Even if we may have a more accurate interpretation, the number of possible interpretations remains unlimited. We still do not know which of the possible interpretations is the intended one unless thorough discussions among the model specifiers and the users are achieved.

2) That the model designer should make a choice among a number of constructs implies that the conceptual model that is eventually constructed is not unique. Moreover, it is not always necessary, desirable or even possible to make such choices. Especially when several local views co-exist or when the perception of the real
world has not yet been mature enough to make such choices.

3) The third reason addressed by Schmid is not clear. What is meant by "interesting result"? To our understanding, interesting result could mean anything that one would like to know. For instance, we might want to know if the conceptual model is consistent. We might want to know if a set of (output) information requirements can be (formally) derived from the conceptual model specification [73]. In this thesis, we show that the consistency of a conceptual model can be formally proved within the framework proposed.

The non-uniqueness of conceptual models with respect to a specification language causes problems:

- Firstly, the "names" that are used to refer to the phenomena in the real world may differ. This situation may happen when several local views coexist and each local view uses a different vocabulary. It can be resolved by equating the vocabularies of the local views.

- Secondly, the concepts or constructs that are being used to represent a phenomenon in the real world may differ. For example, a marriage can be modelled as an entity as well as a relationship [24]. Although the choice between the two can be left to the enterprise administrator [93], we still have to solve the problem of integrating two views, where one of the views models marriage as an entity and the other view sees it as a relationship.

- Thirdly, some application sees one "thing" as an entity as well as a property of another entity. This makes modelling extremely hard [88].

Indeed, there exists no generally accepted solutions to the above problems. Neither in this thesis are we going to resolve them.

2.3 Using Mathematical Logic as a Specification Language

So far to our knowledge, mathematical logic is the only formal setting which offers sound and complete consistency checking mechanisms. This is one reason for choosing to use mathematical logic.

What is mathematical logic?

a) Mathematical logic is a formal language by which one can express and reason about something [Conf. [38]]. Note that we have included in mathematical logic the first order logic, multityped logic, modal logic, intuitionistic logic, dynamic logic, etc..

b) Mathematical logic is a mathematical branch which studies the property of the formal language defined above.

The first statement views mathematical logic as a tool in applied science and the second views mathematical logic as an object for research. The first view of mathematical logic is most often taken by the information modelling scientists (e.g., [18] [20] [84] [22] etc.) The second view is, however taken by mathematicians. In this thesis, we will view mathematical logic as a tool for conceptual model specification and verification.
2.3.1 Advantages

Mathematical logic can be used as a specification language because it is itself a formal language. It offers a uniform and simple representation for all facts. It provides an assertive nature of specification rather than an imperative one. This coincides with current work on specification languages, structured programming formalisms, and denotational theories of programming semantics [102]. It is a truism in the information system area, that a specification should specify WHAT the system is to do rather than HOW to do it [20] [73]. Mathematical logic has proved to be such a formal language which fulfills this goal.

The distinction between WHAT and HOW should not be carried to the extreme. According to Solvberg [90], "the solution of one problem at one level, leads to a requirement for solutions of some new problems at the next level (the HOW of one level is the WHAT of the next level)." Clearly, our distinction of the two is a relative one.

Mathematical logic is powerful enough to express almost everything. It is general enough to be tailored according to the requirements and representation of the informal universe of discourse of the model specifier in exactly the way (out of many) he/she prefers [83].

The other advantage of using mathematical logic to specify a conceptual model is that this framework offers powerful and complete theorem proving mechanisms for deductive reasoning [36] [67] [71]. As we have pointed out in the last two sections, the validity of a conceptual model description cannot be formally checked, we can only formally test the consistency of the specification. Further, the uniqueness of the conceptual model of an application with respect to a modelling approach cannot be guaranteed. This implies that the non-uniqueness of the conceptual model is not due to the use of mathematical logic. In other words, the non-uniqueness of the conceptual model is inherent in the inability of controlling the human perception of reality.

2.3.2 Disadvantages

Nevertheless, there are some problems which associate with the use of mathematical logic to specify a conceptual model:

- First, it has no desirable structure, which makes it unsuitable as a modelling approach. The modelling or structuring task is completely left to the specifier. Two possible solutions to this problem might be:

1) Tailor mathematical logic so that some desirable structure is imposed into the language. That is, the modelling framework provides constructs such as "entity predicate symbols", "relationship predicate symbols", "framework-defined function symbols", "application-defined function symbols", etc. Each class of symbols may have certain pre-defined format. The rules for forming the "wffs" in such a language should also be provided. A set of application independent axioms may also be defined in the framework. To the opinion of this author, CIAM [20] is an example in this approach.
2) Use a separate modelling approach for the modelling task and then specify the conceptual model in terms of a logic oriented language. This is one of the assumptions made in this thesis. As mentioned earlier, one can always tailor logic languages according to one's own requirements and informal representation of reality in exactly the way one likes. This implies that given a piece of reality and a conceptual model which is constructed by using some modelling approach, there is a set of alternatives by which a logic language can be tailored to specify the same piece of reality. The fundamental difference between the language of the modelling approach and the logic language is that some major constraints have been directly incorporated into the modelling constructs. Clearly, different ways of incorporating the major constraints into the constructs result in different classes of modelling approaches. Our conjecture is, that corresponding to each way in which the major constraints are incorporated, there exists exactly one way in which the logic language can be tailored so that the major constraints are listed explicitly. In [83], it is shown that the object association model corresponds to representing an object type by a unary predicate and an association type by an n-ary predicate. The name of the object type and the name of the association type are used as the predicate symbols, respectively.

The second disadvantage of using a logic language is its poor user-interface [51] [85]. This is considered to be one of the limitations of the present framework in this thesis. The solution of the problem relies on further work which should solve the tailoring problem as discussed above. A similar emphasis is taken by [85].

Finally, there is the difficulty of representing fuzzy facts in a logic setting. For instance, the statement "This man is old." is a fuzzy statement, which cannot be easily described as mathematical logic formulae. The present framework is not meant to solve this kind of problems.

2.4 Modelling in the Temporal Dimension

2.4.1 Four Types of Approaches

When temporal dimension is taken into account, there are four types of modelling approaches. Previous studies (e.g., [17] [84] [51]) have shown that the existing approaches can be classified into static (current view/snapshot) approaches and temporal (dynamic/time-perspective) approaches. We classify the existing approaches into four classes: static, dynamic, temporal and full-time perspective approaches.

1) Static approaches which provide facilities only for describing a snapshot of the application. Variants of this type may include process models which can be interpreted as computer instructions. The imperative style of the process model implies a prescription for the software design. In this approach, only one state of
reality is explicitly considered at a time. Static approaches were proposed and focused by the mid-1970's.

2) **Dynamic approaches** which provide facilities for modelling the state transitions without considering in full detail the mechanisms which achieve the transitions. The doctrine of precondition/postcondition is typically used in this type of approaches. Usually, two states are explicitly considered at a time, i.e., the prestate and the poststate. Dynamic approaches started during the late 1970's. Condition/event Petri-nets belong to this type [4]. Other examples are ACM/PCM [16], BASIS [58].

3) **Temporal approaches** which allow the specification of time dependent constraints such as "age must not decrease", etc.. Note that some of these temporal constraints can be specified in terms of preconditions and/or postconditions. However, other temporal constraints cannot be specified in this way, e.g., the frozen price problem demonstrated in [84]. It is assumed in this example that the price to be used for filling a back order is the price at the time when the order was placed. It is further assumed that a back order is filled if a replenishment notice has occurred after the back order was recorded and the quantity ordered can be satisfied. Thus three previous states are referred to in this example: The state in which the order arrived, the state in which the back order was recorded and the state in which the replenishment occurred.

In general, more than two states can be explicitly considered in this type of approach. Temporal approaches begins in the 1980's. Besides [84], other examples of temporal approaches are found in [22] [39], etc.. Temporal logic (modal logic) is used in these approaches. Our framework belongs to this type.

4) **Full time perspective approaches** emphasize the important role and particular treatment of time in modelling reality (see e.g., [19] [20]). Unlike the other three types of approaches, a full time perspective approach gets rid of notations such as states, operations, processes, transactions, etc. CIAM belongs to this class of approaches. Some other research concerning full time perspective is found in [14] [63] [1] [2]. In fact, the number of "states" that is explicitly considered by a full time perspective approach is infinite. Full time perspective approach also begins in the 1980's.

2.4.2 Some Temporal Frameworks

Sernadas models an information system as a network of concurrent processes that exchange messages among themselves and with the environment through the temporal database of the information system [84] [85]. The temporal database is a family of database states, whose indices are made implicit. The processes in the system may only change the current state of the temporal database, and may refer to all of the previous states. A message/process model is proposed in [84] [85]. The message pool is modelled by the relational model which is interpreted as a collection of multityped logic assertions. The processes are described in the INFLOG language which is based upon the modal logic, tense logic and transition logic. The framework is characterized by [84]:
memory independence;
- events and triggers;
- temporal data base;
- updating; and
- privileged present time.

Castilho proposes a temporal framework for database description [22]. The framework consists of a family of specification languages, the individual language IL, the basic temporal language and the temporal language with procedures TL. A database is described in two levels. At the first level, the static constraints and temporal constraints are specified. This level does not consider how the database will be modified. At the second level, a database description includes a set of built-in operations, which are described by their properties. It is by convention that the database transactions are defined in terms of these built-in operations. No higher level constructs are found in the framework for specifying the database transactions. In this respect, we may see that Serndas pays more attention to the description of the processes which correspond to concurrent database transactions [84] [85], while Castilho focus more on the operations upon which the database transactions are defined [22].

In [39], a formal system of modal logic similar to Hoare-style program logic [47] is proposed for specifying databases. Each database instance is defined as a many-sorted algebra where the signature of the algebra constitutes the basis for the database schema. Static integrity constraints are defined as well-formed expressions of the algebra. Transition constraints and update operations are defined as expressions of the modal system. A transaction is proposed to be specified as a sequence of updates. However, this solution is not satisfactory, because we need to specify under what conditions an update operation can take place.

Our framework is similar to the ones proposed in [22], [39] in the sense that operations rather than processes are the building blocks of the specification. The difference is that in our framework the preconditions and postconditions of the operations are explicitly specified instead of being implicit. For example, in [22], a procedure is defined to be a function symbol whose semantics is captured by a set of temporal logic expressions. A similar observation can be made for [39].

In our framework, the effect of an operation is captured by the preconditions and postconditions of the operations, this is similar to most of the dynamic approaches except that those dynamic approaches do not include a temporal assertion in the operation description. We believe that the present research work coincides with the opinion which is held by [6] [19] and [102]. That is, a specification should specify the rules and assumptions explicitly and should suppress exceptional details (when needed), in order to facilitate comprehension and change. To change the description of an operation in the present framework, we need only to change the temporal assertion, the precondition and/or the postcondition of the operation. However, in [22], one needs to change a set of temporal assertions which defines the semantics of the operation. Moreover, the set of temporal assertions that define an operation may spread throughout the specification which makes understanding and change very difficult.
2.5 On Consistency Checking of Specifications

Consistency is one of the qualitative aspects of a product. The meaning of the word consistency is quite different in different fields of science, as pointed out in the introductory chapter.

In this thesis, by consistency of an information system specification we mean that all of the components of the specification are free from conflict. This interpretation of the word coincides with the one found in mathematical logic. The same interpretation is assumed in [18] [20] [62] [85] [15] [5] [22] etc.

The framework for information system specification and verification has been outlined in the introductory chapter. The main points will be repeated here, for the readers convenience.

An information system specification consists of three parts:

1) The static constraint specification is a set of first order logic sentences. The static constraints define the legal states of the information system.

2) The operation specification consists of a set of operation descriptions. Each operation description specifies a historic requirement of the information system to which the operation can be executed. Moreover, an operation description includes a precondition and a postcondition which define the effect of executing the operation.

3) The temporal constraint specification is a set of temporal logic assertions. The temporal constraints define the rules which must be obeyed by any legal sequence of system transitions.

The consistency checking of an information system specification will:

1) Verify that the static constraints are consistent. That is, the set of static constraints must have at least one (mathematical) model. This follows directly from the Extended Completeness Theorem [23].

2) Verify that each of the operations can be executed in a single system state and yields legal system states. This means that prior to executing the operation, there is a legal state in which the precondition of the operation is true, and after executing the operation there is a legal state in which the postcondition of the operation is true.

3) Verify that each sequence of operations that can be executed in the information system must satisfy all the temporal constraints.

2.5.1 Informal Discussion of Verifying Static Constraints

Suppose that in an application, we have the following static constraints:

\[ sc_1 : \text{Every employee earns less than 20000.} \]
\[ sc_2 : \text{Every manager is an employee.} \]
\[ sc_3 : \text{Every manager earns at least 20000.} \]
This set of static constraints is consistent because there is one system state (i.e., a model in mathematical sense) in which there is only one employee whose salary is less than 20000 and there is no manager. Under this interpretation, \(sc_1\) is true because the sole employee earns less than 20000, and there exists no other employees. \(sc_2\) says that every manager is an employee. In our system state there is no manager and hence \(sc_2\) is irrelevant. \(sc_3\) is also irrelevant because we do not care about whether someone earns at least 20000 as there are no managers in this system state. Hence \(sc_2\) and \(sc_3\) are both true in this system state. Another system state could be that there is no employee and manager at all.

Thus, when a database system has been implemented, we are guaranteed to have at least one database state in which the above static constraints are satisfied. Note that in no legal database state may there be any managers recorded. As long as the static constraints are concerned, we are only interested in specifying the legal system states. That is, any database state that satisfies the three static constraints must be considered to be a legal system state. Therefore, the necessary and sufficient condition for having a legal database state is that the static constraints are consistent. Indeed, this necessary and sufficient condition can be seen as a reformulation of the Extended Completeness Theorem in terms of computing terminology.

For the above static constraints, four system states can be constructed. Moreover, the four states are the only system states in which all the static constraints are satisfied. Note that in the following explanation of the satisfiability of the constraints in the states, we have made use of the propositional axiom:

\[ w_1 \rightarrow w_2 \]

if \(w_1\) is not true or \(w_2\) is true, where \(w_1\) and \(w_2\) are propositions. In the following state descriptions, \(a_0\) denotes any individual. A method which formally constructs legal system states will be presented in Sect. 4.8.5.

The four legal states are:

State S0: \(a_0\) is not an employee. \(a_0\) is not a manager. The salary of \(a_0\) is less than 20000. This is a legal state because all the antecedents of \(sc_1\), \(sc_2\) and \(sc_3\) are false, therefore \(sc_1\), \(sc_2\) and \(sc_3\) are true in this state regardless of their consequences. This state corresponds to an empty state.

State S1: \(a_0\) is not an employee. \(a_0\) is not a manager. The salary of \(a_0\) is equal to 20000. This is a legal state because all the antecedents are false, and \(sc_1\), \(sc_2\) and \(sc_3\) are true regardless of their consequences.

State S2: \(a_0\) is not an employee. \(a_0\) is not a manager. The salary of \(a_0\) is greater than 20000.

State S3: \(a_0\) is an employee. \(a_0\) is not a manager. The salary of \(a_0\) is less than 20000. This is a legal state because the antecedents of \(sc_2\) and \(sc_3\) are false and the consequence of \(sc_1\) is true. This state can be considered as a non-empty system state.
Now suppose that we want to impose one more static constraint, due to woman liberation movement, saying that

\[ sc_4: \text{There must be some female who is a manager.} \]

It is easily seen that the new set of static constraints is not consistent. \( sc_4 \) says that there is a female, say Mary, who is a manager. This implies that there is a manager. By \( sc_2 \), Mary earns at least 20000 which is not less than 20000. However, by \( sc_3 \), Mary is an employee since every manager is an employee, and by \( sc_1 \), Mary earns less than 20000. Thus 20000 < 20000 can be inferred, which conflicts the axiom in arithmetic that \((\forall x)(x < x)\).

If a database system for the above application had been implemented, there would exist no database state which satisfied \( sc_1 \), \( sc_2 \), \( sc_3 \), and \( sc_4 \). That is, we could not enter any information about an employee or about a manager without violating one of the constraints. In practice, if this situation arises (due to failure of discovering the inconsistency of the specification), then we must re-examine the specification as well as the design and implementation to remove the possible bugs and re-test the system again. Clearly, this is a costly process.

### 2.5.2 Analysis of Operation Descriptions

The consistency checking of the static constraints enables us to ensure that there exists at least one system state which satisfies the static constraints. However, this kind of consistency checking is not enough. An information system is an evolutionary object which involves inserting, deleting and updating the data which is stored in the system. The evolution aspect of an information system must somehow be specified in a specification. The evolution of an information system can be specified in many ways. In a static framework, the evolution of an information system is usually specified by a set of high level procedures which can be interpreted as a collection of instructions of a machine. The semantic constraints and/or evolution information are hidden in the procedure specifications [20] [21]. In a dynamic framework, the evolution is usually specified in terms of state transitions which is achieved by specifying the preconditions and postconditions of the transitions. In a temporal framework, the evolution can either be specified by precondition/postcondition as we do in this thesis, or by a set of temporal assertions which together describe the transition from one state to another. The latter strategy is used in [84] [85] [22] and [39]. In a full time perspective framework, the evolution is specified in terms of a set of rules with time arguments which explicitly and declaratively describe the evolution of the system in the course of time. Note that no state transition is implied in a time perspective framework, instead, the system’s behaviour along the time course is described.

Suppose that in addition to the static constraints \( sc_1 \), \( sc_2 \) and \( sc_3 \), we have the following operations to be performed in the information system for the application in question:

Operation "hire an employee a0", abbreviated as hire(a0), which states that in a state in which a0 is not an employee, then after performing the operation hire(a0) the system enters a state in which a0 is recorded as an employee whose salary is less than 20000 (see the figure below).
operation: hire(a0)

temporal

assertion: not defined yet.

precondition: a0 is not an employee.

postcondition: a0 is an employee with

salary less than 20000.

end:

Operation "fire an employee a0", abbreviated as fire(a0), which states that in a state in which a0 is an employee but not a manager, then after performing the operation fire(a0) the system enters a state in which a0 is not an employee. That is, a0 is deleted from the list of employees:

operation: fire(a0)

temporal

assertion: not defined yet.

precondition: a0 is an employee and

a0 is not a manager.

postcondition: a0 is not an employee.

end:

Operation "promote an employee a0 to be a manager", abbreviated as promote(a0), which states that in a state in which a0 is an employee but not a manager, then after performing the operation promote(a0) the system enters a state in which a0 becomes a manager:

operation: promote(a0)

temporal

assertion: not defined yet

precondition: a0 is an employee and

a0 is not a manager.

postcondition: a0 is a manager.

end:

The above operation descriptions specify the effects of the operations upon the system states. If we compare them with the legal system states S0, S1, S2, S3, we see that the operation hire(a0) can be performed in states S0, S1, S2 but not in state S3. This is because in states S0, S1 and S2, a0 is not an employee which satisfies the "applicability condition" of the operation [15]. While in state S3, a0 is already an employee and the "applicability condition" of the operation is not satisfied. After performing the operation hire(a0), the system enters state S3, since this state is the only state in which a0 is an employee. Thus, the analysis of the operation hire(a0) results in the following state transitions:

```
S0  hire(a0)  S3
   ।                     ।
S1  hire(a0)  S3
   ।                     ।
S2  hire(a0)
```
However, among the three transitions, only the transition from state S0 to state S3 is desirable. This is because that in state S0, one knows that the salary of a0 is less than 20000 and the hire operation does not change this information. This means that anything that is not changed by the operation should be preserved. This is called the "frame problem" in artificial intelligence [65] [32] [72] [50]. The salary of a0 in states S1 and S2 is not less than 20000, the transitions from state S1 to state S3 and from state S2 to state S3 will change the salary information. Thus, the analysis of the hire operation finally leads to the following figure where the question mark denotes an illegal state.

\[ S0 \xrightarrow{\text{hire(a0)}} S3 \]
\[ S1 \xrightarrow{\text{hire(a0)}} ? \quad \text{(illegal state)} \]
\[ S2 \xrightarrow{\text{hire(a0)}} \]

Note that the above discussion has not presupposed any particular individual a0. This means that the (static) consistency checking and the analysis of the hire operation is valid for all individuals though we choose a0 as the representative.

In a similar way, we can analyse the other two operations. The analysis results in the state transition diagram below:

\[ S0 \xrightarrow{\text{hire(a0)}} S3 \]
\[ S1 \xrightarrow{\text{fire(a0)}} S2 \]
\[ S0 \xrightarrow{\text{promote(a0)}} S3 \]
\[ S1 \xrightarrow{\text{hire(a0)}} ? \quad \text{hire(a1)} \]
\[ S2 \]

**Fig. 1. A state transition diagram for the example**

Fig.1 shows that there is no chance for the promote operation to be performed, since its performance always leads to an illegal state. This is due to the static constraints, which do not permit any manager to be recorded.

\[ S0 \xrightarrow{\text{hire(a0)}} S3 \]
\[ S0 \xrightarrow{\text{fire(a0)}} \]

**Fig. 2. The transition diagram after removing illegal transitions**

If we remove all the transitions which lead to illegal states, then the resulting figure indicates that only two of the operations, i.e., hire and fire, can ever be performed (Fig.2).
The analysis indicates that we must re-specify the satatic constraints as well as the operation descriptions so that all the operations that are needed to be performed in the future information system can ever be performed.

2.5.3 Consistency Checking of Temporal Constraints

To illustrate the importance of the consistency checking of the temporal constraints, we suppose that the application as described above has a temporal constraint saying that "Whoever has been an employee (probably has been fired) cannot be hired again". This temporal constraint excludes any sequence of operations of the form

...hire(a0)...hire(a0)... 

That is, any sequence of operations cannot contain more than one hire operation for any given individual. The analysis of the operation descriptions in the last subsection dealt with only one operation at a time. It failed to reveal the property of a sequence of operations. As can be seen from Fig.2, repeatedly executing the operations hire(a0), fire(a0) results in the sequence hire(a0), fire(a0), ..., hire(a0), fire(a0), ... This sequence is obviously unacceptable by the temporal constraint above. This suggests that there must be some additional information which should be associated with the description of the operations so that undesirable sequences of operations can never be performed. In the example above, this additional information that should have been associated with the hire operation will refer to the history of the system which states that always in the past, a0 was not an employee. We will show in Chapter 3 how this kind of information can be specified, and in Chapter 4, how the temporal constraints can be checked.

The above discussion serves to show the importance of consistency checking concerning the static and temporal constraints as well as the operation descriptions. It also illustrates the meaning of consistency in this thesis. In what follows, we try to relate the above discussion to some existing work in the information systems field and the database field.

2.6 Existing Works in Consistency Checking

2.6.1 Consistency Checking in Information System Field

In the information systems field, the concept of consistency is well-defined. That is, a specification is considered to be consistent if the components of the specification contains no conflict. Work relating to the consistency checking of information system's specifications is found in [62], [64], [58], [5], [73], [74] and [92].

In [62] and [64], the specification of an information model is based upon predicate logic. Consistency checking of the information model with respect to an information base is discussed. A consistency checking system IMT (Information Modelling Tool) is proposed which uses the resolution principle as its proof mechanism. The method is effective in detecting the inconsistency of the specification but it
is generally not able to conclude that the specification is consistent [64].

In [5], a framework for specifying the objects of an information system is proposed. The objects include processes, signals and information. The relationships between these objects are that information may be related to and/or produced by processes, signals may initiate and/or be effected by processes, information may exist at a moment and signals occur at a moment, processes proceed in an interval, etc. A set of axioms for all applications is maintained by the consistency checking system which takes a specification of any particular application and infer upon the axioms and the specification. Inconsistent specifications can therefore be detected by the system. 54 examples are shown in the study, each of which is detected to be inconsistent. Again, the study does not claim to be able to determine the consistency of a specification [5].

In [58], a Behavioral Approach to the Specification of Information Systems (BASIS) is proposed. As a part of the approach, the specification of abstract object classes concerns the consistency issue. An abstract object class consists of an abstract image, an abstract invariant and a list of operation descriptions. The abstract image of an object is a list of the attributes associated with the objects along with constraints on the value set of these attributes. An operation is specified by a pair of one precondition and one postcondition which characterize the effect of the operation. The abstract invariant of an object class defines the inter-relationships between the components of the abstract image. It is possible to eliminate the invariant by incorporating it into the preconditions and postconditions of the operations.

A Hoare-style consistency checking framework is used in verifying the specification of the object classes. That is, the consistency checking proves that the create operations establish the invariants and then show that each update operation preserves the invariants. An existing theorem prover is used in the BASIS approach for this purpose [12] [58]. One of the problems of the BASIS approach is that inter-object-class constraints cannot be specified while in our approach this can easily be done. Another problem of the BASIS approach is that temporal constraints are not considered.

In [73], the notion of consistency is assumed to have a different meaning from the one that is held in this thesis. The verification of a specification is to prove that the information requirements of an information system can be formally derived from the specification. Two methods are proposed, the one which uses the information precedent relation [56] and the one which uses process grouping. A recursive procedure which implements the precedent relation approach is found in [92].

2.6.2 Consistency Checking in the Database Field

Within the data base field, the notion of consistency and consistency checking assume quite different meanings among the researchers. For example, in [30] and [94] the following four qualitative aspects are addressed:
1) **Security** deals with preventing users from accessing and modifying the data in unauthorized ways.

2) **Consistency** deals with preventing semantical errors which may arise due to the interaction of two or more processes operating concurrently on shared data.

3) **Reliability** deals with preventing errors due to the malfunctioning of system's hardware or software.

4) **Integrity** deals with the prevention of semantical errors made by (authorized or not authorized) users due to their carelessness or lack of knowledge.

Here, we see that "consistency" deals with **concurrency control**. Integrity, on the other hand, can be seen as the implementation of the semantic constraints (static and temporal constraints) of an information system's specification.

In [83], **integrity constraints** are also called consistency constraints. A detailed discussion about whether to treat integrity constraints as a part of a conceptual model is found in this paper. The conclusion is that a conceptual model should include both a (structured) model and its integrity constraints.

In [15], a database schema definition language Beta is proposed. A B-calculus (Beta-calculus) framework is used to verify the schema description. A data base schema is said to be consistent if it does not contain conflicting statements. The verification of the schema specification is to prove that the schema description is consistent. In this respect, the notion of consistency is in agreement with ours.

The verification process in [15] consists of two parts. First, the schema specification is verified statically with respect to a set of axioms, which are defined for all applications. Second, those properties that cannot be verified statically are verified by using a particular database. According to [15], "the verification method based on predicate calculus and mathematical semantics have well known theoretical limitations". However, these limitations are not exemplified, and there is no support for the statement that can be found in other papers written by Brodie.

### 2.6.3 Semantic Constraints and Integrity Constraints

In the introductory chapter of this thesis, we mentioned that an information systems specification should contain a conceptual model, in some formal language, which depicts a part of reality that is interest to the application in question. Thus, by semantic constraints we mean those statements that are specified to reflect the laws of the application such as static constraints and temporal constraints. An information system is usually supported by a data base system which is specified by a data base schema using a data model. Thus, by integrity constraints we mean those statements of a database schema that are used to express those properties of the data which cannot be expressed conveniently by the data model of the data base system.

Integrity constraints are not necessarily identical to the semantical constraints that are specified in the specification. Design decisions must be made concerning which concepts or constructs of the data
definition language will be used to implement a semantic constraint. For example, a semantic constraint saying that "the stock level of item A should not be lower than 200 pieces" can be implemented in several ways. It can be implemented as some special code written in some procedure or as an integrity constraint which must be evaluated each time a delivery of the item is to be issued. In a relational database system, the above semantic constraint can also be implemented by specifying that the domain of the stock-level attribute of item A is within some range whose lower bound is 200. In this case, no integrity constraint is needed for the semantic constraint.

In the specification phase, there exists no implemented data base that satisfies the specification. So, what we can do is to formally (or informally) verify the specification. The verification is supposed to reveal syntactical as well as semantical errors of the specification. The discovery of these errors in the specification phase hopefully will reduce the total development cost (conf. [26]).

2.6.4 Consistency Checking and Integrity Enforcement

It is useful to distinguish between consistency checking of information system specifications and integrity enforcement of databases. Consistency checking is performed at a logical level. The purpose is to make sure that the specification is free from conflict. Thus, the implementation of a database system according to a verified specification will be free from logical errors. Consistency checking is performed once, when the specification, which acts as a documentation of the system, is changed. On the other hand, integrity constraints are enforced each time an updating of the data base takes place.

The process of ensuring that the database content does not violate any of the integrity constraints at the point of updating a database is called integrity enforcement. During the last decade, many integrity enforcement techniques have been proposed [94] [95a][44] [11] [45] [82] [54] [69] [70] [30]. It is commonly recognized that the integrity enforcement process is often costly and time consuming. Most of the proposed techniques focus on making integrity enforcement more efficient. A commonly used technique is to enforce a subset of the integrity constraints at updating because an updating usually affects only certain integrity constraints. Another technique is to delay the enforcement until the data is used [54]. It is also possible to retain database integrity periodically in some cases. However, a cost-benefit solution for enforcing the database integrity has not been found. That is, we sometimes just have to tolerate certain "inconsistency" in a database for cost-benefit considerations. This situation seems to reduce the potential value of consistency checking of a specification, because the "consistency" of the database cannot always be maintained. What is the reason for ensuring the consistency of the specification?

To clear this cloud, we cite some of the existing works in the following. Without loss of generality, we make no distinction between an information system specification and a database schema description.

- According to Nicolas and Gallaire, it is in fact implicitly followed in relational data base theory that the data base content is considered as an interpretation of a first order theory T which is the data base schema. "The world ought to be a (mathematical) model of the theory T. How can this be ensured? In fact there is
no way to ensure this. The only thing that can be done is to verify the consistency of the theory. If it is not consistent, no model of it exists and so, the world cannot be one model either." [68]

According to Brodie, "a database exhibits semantic integrity only if all specified constraints are satisfied. If any two of the large number of constraints are inconsistent, no database can exhibit semantic integrity." [15].

In fact, two things must be distinguished: the quality of a specification and the ability of supporting the specification. As discussed above, the consistency of a specification is one of the necessary conditions for correctly modelling the reality [69]. On the contrary, the requirement of supporting the specification implies that more sophisticated integrity enforcement techniques should be developed.
CHAPTER 3:

THE FRAMEWORK FOR SPECIFYING
INFORMATION SYSTEMS
3 THE FRAMEWORK FOR SPECIFYING INFORMATION SYSTEMS

As mentioned in the introductory chapter, the framework for specifying information systems consists of three parts: the specification of the static constraints, the specification of the operations and the specification of the temporal constraints.

The functions and the inter-relationships of these components are shown in the following figure (Fig.3).

---

LEGEND:

**S_{jk}**  \( k = 1, \ldots, n \), an arbitrary legal system state.

**op_{ik}**  \( k = 1, \ldots, n-1 \), an operation which can be performed in state **S_{jk}** and result in state **S_{jk+1}**.

**A** A binary relation between A and B such that A satisfies B. E.g., if

**satisfies:** A is a legal system state and B is a static constraint, then the arrow from A to B indicates that B holds in A.

**Fig. 3.** The overview of the framework

---

Fig. 3 shows that the static constraints should be satisfied by any legal system state. The legal system states range over the time axis which is implicit in the framework. The transition from one legal system state to another is caused by the execution of an operation. Clearly, only those operations that go from legal states to legal states are well-defined with respect to state transitions. That is, a well-defined operation with respect to state transitions maps a legal state to another legal state. In practice, we are only interested in specifying well-defined operations. Because the transition from a legal state to an illegal state indicates that the transition will violate some of the static constraints. This in turn indicates that the database content resulting from such a transition will not
represent the perceived real world. Thus, the analysis of the operation descriptions should reveal the logical and semantical errors which exist in the information systems specification.

The above discussion only deals with individual operations. For sequences of operations, Fig. 3 shows that any legal sequence of operations must satisfy the temporal constraints. In other words, the temporal constraints specify some restriction on sequences of operations such that the evolution of the information system will at all times obey the perceived evolution of the application. Rather than considering only two consecutive states, the introduction of the temporal dimension enables us to consider sequences of states.

3.1 Static Constraints in a First Order Language

3.1.1 The First Order Language

The first order language is used in this thesis for specifying the static constraints. The reason for using such a language is that existing proof method such as the resolution principle can be used. We will later make use of the unification algorithm to construct a unifiability digraph by which we can determine the consistency of a set of formulae very effectively (see Sect. 5.3).

The first order language $L$ can be defined as follows:

I) The alphabet of the language contains

   a) A set of non-logic symbols which is partitioned into three groups: predicate symbols, function symbols and constant symbols.

   b) A set of logic symbols

      - parentheses ),( 
      - variables $u$, $v$, $x$, $y$, $z$, ...
      - connectives $\neg$ (not), $\&$ (and)

      For simplicity, we include $\lor$ (or) and $\rightarrow$ (implication) as abbreviations which are to be defined latter.

      - quantifier $\forall$ (for all)

      We include $\exists$ as an abbreviation which will be defined later; and

      - identity $\equiv$.

II) The grammar of the language contains the following rules
a) The rules for forming the terms of the language $L$:

1) A variable is a term of $L$.

2) A constant symbol is a term of $L$.

3) If $f$ is an $m$-ary function symbol of $L$ and $t_1$, ..., $t_m$ are terms of $L$, then

$$f(t_1, ..., t_m)$$

is a term of $L$.

4) A string of symbols is a term of $L$ only if it can be shown to be a term of $L$ by a finite number of applications of 1)—3).

b) The rules for forming the atomic formulae of $L$:

1) If $t_1$ and $t_2$ are terms of $L$, then $t_1 = t_2$ is an atomic formula of $L$.

2) If $P$ is an $n$-ary predicate symbol of $L$ and $t_1$, ..., $t_n$ are terms of $L$, then $P(t_1, ..., t_n)$ is an atomic formula of $L$.

c) The rules for forming the well-formed formulae (wffs) of $L$:

1) An atomic formula is a wff.

2) If $w_1$ and $w_2$ are wffs, then $\neg w_1$, $(w_1 \land w_2)$ are wffs.

3) If $w$ is a wff, and $x$ is a variable, then $(\forall x)w$ is a wff.

4) A sequence of symbols is a wff only if it can be shown to be a wff by a finite number of applications of 1)—3).

The logic symbols $\lor$, $\rightarrow$ and $\exists$ are defined as

- $(w_1 \lor w_2)$ is the abbreviation of $\neg(\neg w_1 \land \neg w_2)$.
- $(w_1 \rightarrow w_2)$ is the abbreviation of $(\neg w_1 \lor w_2)$.
- $(\exists x)w_1$ is the abbreviation of $\neg(\forall x)\neg w_1$.

where $w_1$ and $w_2$ are wffs.

A sentence is a wff with no free variables. A set of sentences of a language $L$ is called a theory of $L$ (see e.g., [23] [20]). For simplicity, we will very often use formulae or wffs as indistinguishable from sentences, when no confusion can arise.
3.1.2 The First Order Axioms

The logical axioms for the first order language are divided into three groups:

1) **Semantical Axioms**: Every formula \( w_2 \) of \( L \) which can be obtained from a tautology \( w_1 \) of the propositional calculus by (simultaneously and uniformly) substituting formulae of \( L \) for the propositional symbols of the propositional calculus is a logical axiom for \( L \).

2) **Quantifier Axioms**:

   a) If \( w_1 \) and \( w_2 \) are formulae of \( L \) and \( x \) is a variable not free in \( w_1 \), then the formula

   \[ (\forall x)(w_1 \rightarrow w_2) \rightarrow (w_1 \rightarrow (\forall x)w_2) \]

   is a logical axiom.

   b) If \( w_1 \) and \( w_2 \) are formulae and \( w_2 \) is obtained from \( w_1 \) by freely substituting each free occurrence of \( x \) in \( w_1 \) by the term \( t \) (i.e., no variable \( y \) in \( t \) shall occur bound in \( w_2 \) at the place where it is introduced), then the formula

   \[ (\forall x)w_1 \rightarrow w_2 \]

   is a logical axiom.

3) **Identity Axioms**: Let \( u, v \) be variables, \( t(x_0, \ldots, x_n) \) be a term and \( w(x_0, \ldots, x_n) \) be an atomic formula. The formulae

   - \( u=u \)
   - \( u=v \rightarrow t(x_0, \ldots, x_{i-1}, u, x_{i+1}, \ldots, x_n) = t(x_0, \ldots, x_{i-1}, v, x_{i+1}, \ldots, x_n) \)
   - \( u=v \rightarrow \neg w(x_0, \ldots, x_{i-1}, u, x_{i+1}, \ldots, x_n) \rightarrow w(x_0, \ldots, x_{i-1}, v, x_{i+1}, \ldots, x_n) \)

   are logical axioms.

As inference rules, we use

1) **Rule of Detachment** (or *Modus Ponens*):

   From \( w_1 \) and \( w_1 \rightarrow w_2 \) infer \( w_2 \).

2) **Rule of Generalization**:

   From \( w \) infer \( (\forall x)w \).

Following standard usage, we use \( \vdash w \) to mean that \( w \) is a theorem of \( L \). If \( W \) is a set of sentences of \( L \), then \( W \vdash w \) means that there is a proof
of \( w \) from the logical axioms and \( W \). That is, there is a sequence of applications of the inference rules which derives \( w \) from the logical axioms and \( W \). \( W \) is inconsistent iff every formula of \( L \) can be deduced from \( W \). Otherwise, \( W \) is consistent.

### 3.1.3 The Concepts of Structure and Model

In this section, we will use the word model in the mathematical sense. The reason is to keep the presentation as it is in mathematics. Note also the word structure which is used in this section: the meaning of a structure here is completely different from the one that is commonly used for data structure, process structure in computer science.

The concept of a structure for a formal language \( L \) is a pair \( S(U, I) \) defined as follows:

a) \( U \) is a non-empty set of individuals called the universe.

b) \( I \) is a mapping between the non-logic symbols of \( L \) and the elements of \( U \). \( I \) is defined as follows.

1) \( I \) assigns an individual \( e_i \) of \( U \) to each constant symbol \( c_i \) of \( L \). That is, \( e_i \) is called the interpretation of \( c_i \) in the structure. In this thesis, we will only deal with one structure at a time. Therefore, we may denote the fact that the interpretation of \( c_i \) is \( e_i \) as \( I(c_i) = e_i \). This convention is similarly applied below.

2) \( I \) assigns an \( n \)-ary mapping \( m : U^n \rightarrow U \) to each \( n \)-ary function symbol \( f \) of \( L \), where \( U^n \) is the \( n \)-ary Cartesian product of \( U \). That is, \( I(f) = m \).

3) \( I \) assigns an \( n \)-ary relation \( R \subseteq U^n \) to each \( n \)-ary predicate symbol \( P \) of \( L \). That is, \( I(P) = R \).

Note that the mapping defined in Sect.2.1.3 mapped from some part of reality to a formal language. However, the interpretation \( I \) is defined from the formal language to some possible world. It is true that there usually exist an infinite number of structures for a given language. This is because that we can interpret the non-logic symbols of a formal language in a number of different ways as we have informally shown in Sect.2.1.4.

### 3.1.4 The Notion of Satisfaction

Given a set \( W \) of wffs in \( L \), we can define a structure for the set \( W \). In order to do this, we define \( L/W \) to be the language whose non-logical symbols consists of only those non-logical symbols that appear in \( W \). A structure for the set \( W \) of wffs is a structure of \( L/W \). If every wff in \( W \) is true in the structure, then we say that the structure is a model (in the mathematical sense) for the set \( W \) of wffs.

Let \( S \) be a structure and \( W \) a set of wffs of \( L \), respectively. Further, let \( w \) be a wff in \( W \). If \( S \) is a model of \( W \), then we denote this fact symbolically as \( S \models W \), which means that every wff in \( W \) is true in the
structure. We also use $S \models w$ to denote that the wff $w$ is true in $S$.
The following phrases will be used to mean the same thing in this thesis:

- $w$ holds in $S$;
- $S$ satisfies $w$;
- $w$ is satisfied by $S$;
- $S$ is a model of $w$;
- $w$ is true in $S$;
- $S \models w$;
- $w$ is valid in $S$.

where $w$ is a wff and $S$ is a structure of $L$.

A sentence $w$ is valid iff it is valid in every structure of $L$. This notion will be denoted as $\models w$.

A wff $w_2$ is a consequence of another wff $w_1$, in symbols $w_1 \models w_2$, iff every model of $w_1$ is a model of $w_2$. A wff $w$ is a consequence of a set of wff $W$, in symbols $W \models w$, iff every model of $W$ is a model of $w$. This follows that

$$(W \cup \{w_1\}) \models w_2 \iff W \models (w_1 \rightarrow w_2)$$

where $w_1$ and $w_2$ are wffs.

We may now formally define the notion of satisfiability.

Let $S(U, I)$ be a structure of $L$. Let $t(x_0, \ldots, x_q)$ denote a term $t$ whose variables form a subset of $\{x_0, \ldots, x_q\}$. Similarly, let $w(x_0, \ldots, x_q)$ denote a formula $w$ whose free variables form a subset of $\{x_0, \ldots, x_q\}$. Further, let $t[e_0, \ldots, e_q]$ denote the value of $t(x_0, \ldots, x_q)$ when $x_0, \ldots, x_q$ are taken to be $e_0, \ldots, e_q$, respectively, where $e_0, \ldots, e_q$ are in $U$. We define $t[e_0, \ldots, e_q]$ as follows:

i) If $t$ is $x_i$, then $t[e_0, \ldots, e_q]$ is $e_i$.

ii) If $t$ is a constant symbol $c$, then $t[e_0, \ldots, e_q]$ is $I(c)$.

iii) If $t$ is $f(t_1, \ldots, t_n)$, where $f$ is an $n$-ary function symbol, then $t[e_0, \ldots, e_q]$ is $m(t_1[e_0, \ldots, e_q], \ldots, t_n[e_0, \ldots, e_q])$, where $m = I(f)$.

For brevity, we write $S \models w[e_0, \ldots, e_q]$ for $e_0, \ldots, e_q$ satisfies wff $w$ in structure $S$. We have

i) Suppose that $w(x_0, \ldots, x_q)$ is the atomic formula $t_1 = t_2$, where $t_1(x_0, \ldots, x_q)$ and $t_2(x_0, \ldots, x_q)$ are terms. Then,

$S \models (t_1 = t_2)[e_0, \ldots, e_q]$ iff

$t_1[e_0, \ldots, e_q] = t_2[e_0, \ldots, e_q]$. 
ii) Suppose that \( w(x_0, \ldots, x_q) \) is the atomic formula \( P(t_1, \ldots, t_n) \), where \( P \) is an \( n \)-ary predicate and \( t_1, \ldots, t_n \) are terms whose variables are among \( x_0, \ldots, x_q \). Then
\[
S \models P(t_1, \ldots, t_n)[e_0, \ldots, e_q] \iff R(t_1[e_0, \ldots, e_q], \ldots, t_n[e_0, \ldots, e_q])
\]
where \( R = I(P) \).

Suppose that \( w(x_0, \ldots, x_q) \) is a formula of \( L \):

i) If \( w \) is \( w_1 \& w_2 \), then \( S \models w[e_0, \ldots, e_q] \iff S \models w_1[e_0, \ldots, e_q] \) and \( S \models w_2[e_0, \ldots, e_q] \).

ii) If \( w \) is \( \neg w_1 \), then \( S \models w[e_0, \ldots, e_q] \iff \neg S \models w_1[e_0, \ldots, e_q] \).

iii) If \( w \) is \( (\forall x_i) w_1 \), where \( i \leq q \), then \( S \models w[e_0, \ldots, e_q] \iff \) for every \( e \) belongs to \( U \), \( S \models w_1[e_0, \ldots, e_{i-1}, e, e_{i+1}, \ldots, e_q] \).

iv) This definition is extended to \( V, \; \rightarrow \; \) and \( I \), in a similar way.

The above definition is adapted from [23].

### 3.1.5 First Order Theory and Conceptual Model

A result from model theory seems very useful for our purpose which says that a set of wffs is consistent iff it has a model, i.e., the Extended Completeness Theorem [23] [9]. As mentioned earlier, a theory of a language \( L \) is a set of sentences of \( L \). By virtue of the Extended Completeness Theorem, a theory \( T \) is consistent iff it has a model. That is, \( T \) is consistent iff there is a structure \( S \) in which every sentence \( w \) of \( T \) is true.

If a conceptual model of an application is specified in terms of the first order language, then the conceptual model description can be seen as a theory of the application. This view is essentially the same as the one taken by Bubenko [20] [21]. In this context, an important aspect for a conceptual model is that its theory \( T \) should be consistent. For otherwise, there will exist no structure \( S(U, I) \) which can be a model of \( T \). That is, no interpretation of the symbols used in \( T \) makes every sentence of \( T \) true in the structure.

In our framework, a conceptual model consists of three parts: a set of first order sentences which describes a snapshot of the application, a set of operation descriptions and a set of temporal constraints. The set of first order sentences can be seen as a sub-theory of the application. As discussed in Sect.2.5.1, this set of sentences specifies the legal system states. According to [69], a necessary (but not sufficient) condition for considering the sub-theory to be a "conceptual model" of a snapshot of a part of reality is that the sub-theory should be consistent. That is, the sub-theory should have a (mathematical) model. Usually, there are a number of models for a consistent theory. Each model corresponds to a state of affairs under the theory (see Sect.4.8.5).
An information system is an implementation of a conceptual model. If the conceptual model describes a snapshot of a piece of reality, it follows that a state of affairs of reality can be seen as a model for the theory of the conceptual model. This is essentially the second approach addressed in [68]. By the Extended Completeness Theorem, if the conceptual model is inconsistent, then no state is able to satisfy the conceptual model description. This problem of unsatisfaction exists there even if we do not discover it.

3.1.6 A Simple Example

As an example, consider the following three static constraints:

\[ SC_1: \text{Every employee earns less than 20000.} \]
\[ SC_2: \text{Every manager is an employee.} \]
\[ SC_3: \text{Every manager earns at least 20000.} \]

In order to formally specify these static constraints, we define the following non-logical symbols:

- The constant symbol \( 20000 \).
- The unary predicate symbol \( E(x) \) which represents that \( x \) is an employee.
- The unary predicate symbol \( M(x) \) which represents that \( x \) is a manager.
- The binary predicate symbol \( \text{EARNS}(x,y) \) which represents that \( x \) earns \( y \).
- The binary predicate symbol \( \text{LT}(x,y) \) which represents that \( x \) is less than \( y \).
- The binary predicate symbol \( \text{GE}(x,y) \) which represents that \( x \) is greater than or equal to \( y \).

The three static constraints can be formally specified as:

\[ SC_1: (\forall x)(\forall y) (E(x) \& \text{EARNS}(x,y) \rightarrow \text{LT}(y,20000)) \]
\[ SC_2: (\forall x) (M(x) \rightarrow E(x)) \]
\[ SC_3: (\forall x)(\forall y) (M(x) \& \text{EARNS}(x,y) \rightarrow \text{GE}(y,20000)) \]

3.2 The Specification of Temporal Constraints

3.2.1 The Temporal Language

In this thesis, we will only consider a limited temporal language for expressing the temporal constraints. The reason is to simplify the consistency checking of the specification. In this sense, the present framework is not powerful enough nor complete from a practical point of view.
In the formal definition of the temporal language, we use the following abbreviations:

- ta —— temporal assertion
- tap —— temporal assertion to the past
- taf —— temporal assertion to the future
- tqp —— temporal quantifier to the past
- tqf —— temporal quantifier to the future

In BNF, the temporal language can be defined as follows:

<temporal-constraint> ::= <ta>
	<ta> ::= ¬<ta> | <ta1> & <ta2> | <tap> | <taf> | <wff'>
		| EXECUTABLE(<op>)
	<tap> ::= <tqp><wff'> | <tqp><tap> | ¬<tap> | <tap1>&<tap2>
	<taf> ::= <tqf><wff'> | <tqf><taf> | ¬<taf> | <taf1>&<taf2>
	<tqp> ::= always+ | always- | ever+ | ever-'
	<tqf> ::= always+ | always- | ever+ | ever-'
	<wff'> ::= wff in which free variables are <parameters>s.
	<parameter> ::= x | y | .. | <function-symbol>(<parameter-list>)
	<parameter-list> ::= <parameter> | <parameter>,<parameter-list> |
	<function-symbol> ::= any function symbol of the first order language L.

The above definition is extended to include V and → as usual.

EXECUTABLE(<op>) is an abbreviation of three conditions of the information system concerning the application of the operation <op>. The semantics of the temporal assertions are to be defined in Sect.3.2.3.

3.2.2 On Parameters

In agreement with [39], we distinguish global and local quantifications. The distinction is made by introducing the concept of parameters into the temporal constraints and the preconditions and the postconditions of the operations. While a universally quantified variable (e.g., ∀x) ranges over all the individuals of a specific state, a parameter (e.g., x) is allowed to range over all the individuals of all the system states. That is, a parameter is allowed to take all the possible values so long it assumes only one value at a time.

The concept of parameter here resembles the concept of exogenous variables in some economic theory. An exogenous variable such as the price of a commodity is assumed to be constant within some period of
time although it may change for period to period. On the other hand, a universally quantified variable in first order logic resembles the concept of an *endogenous variable* in the economic system, which changes within a period of time. For example, the *budget line equation* for any individual in a closed market containing two commodities can be represented as follows:

\[ P_1 x_1 + P_2 x_2 = B \]

where for \( i=1,2 \), \( P_i \) is the price of commodity \( i \), \( x_i \) is the the quantity of commodity \( i \) that is purchased and \( B \) is the budget of the individual. Clearly, for any given person and during any given period of time, \( P_1 \) and \( B \) are constant. The more one purchases commodity 1, the less he/she can purchase commodity 2 and vice versa. Thus, \( x_1 \), \( x_2 \) are variables in a given state (i.e., a given period). However, prices of commodities may change from period to period. It is assume that the prices of commodities 1 and 2 are determined exogenously according to some exogenous factors such as the increase of production costs. However, price changes in such a model do not subject to the budget line equation because what the equation constrains are:

1) The quantities of purchasing \( x_1 \) and \( x_2 \) must satisfy some linear relation.

2) The total amount of money that can be spent in purchasing the commodities must not exceed the budget available.

3) All the budget must be spent, i.e., the *equilibrium assumption*.

In order to study the effect of price changes on the consuming behavior of any given individual, we must allow \( P_i \) to change from period to period. Thus, \( x_1 \) and \( x_2 \) can only assume values in a "legal" state, which satisfies the budget line equation, whereas \( P_1 \) and \( P_2 \) may assume their values in a global sense. That is, the budget line equation cannot constrain the values that are assigned to \( P_1 \) and \( P_2 \) in any given state. (see the figure below).

![Fig. 4. The diagram for the budget lines](image)

In Fig. 4, curve 1 is the budget line for period 1 during which commodity 1 has price \( P_1 \) and commodity 2 has price \( P_2 \). The endogenous (local) variable \( x_1 \) may range from 0 to \( B/P_1 \) and \( x_2 \) may range from 0 to \( B/P_2 \). All the points \((x_1, x_2)\) on curve 1 satisfies the budget line constraint during period 1. During period 2, the price of commodity 2 has increased to \( P_2 + \Delta P_2 \). This does not subject to the constraint imposed by the budget line equation. The change of \( P_2 \) is a global change, for example, due to an increase of the price of some raw material which is used to produce \( x_2 \). Curve 2 shows that the amount of
commodity 2 that can be consumed by the individual will decrease. That is, the local variable $x_2$ will range from 0 to $B/(P_2+\Delta P_2)$.

In fact, there is some constraint upon the parameters $P_1$, $P_2$, e.g., $P_1 > 0$ and $P_2 > 0$. If the quantities $x_1$, $x_2$ cannot be less than one, then $P_1 < B$ and $P_2 < B$ will be another constraint upon the exogenous variables. However, the first constraint is not defined by the budget line equation but by the knowledge that prices cannot be negative. For instance, we could have assigned a negative value to $P_1$ and a positive value to $P_2$ and the budget line equation could still have some points $(x_1, x_2)$ which satisfy it.

The exogenous variables $P_1$ and $P_2$ may subject to some "temporal constraint" such as the price change of any commodity cannot be greater than 5% from period to period. That is,

$$\Delta P_1/\Delta t < 5\% \text{ per } \Delta t$$

The distinction of local and global quantifications is also found in [84] where a constant domain assumption is made in order to introduce

$$(\forall x)(\text{always} w) \longrightarrow \text{always} (\forall x) w$$

as an axiom (i.e., the Barcan Formula). It is beyond the scope of this thesis to carry out a comparison here.

3.2.3 The Semantics

Let $\sigma = \ldots S_1 S_0 S_1 S_2 \ldots$, denote a sequence of states, where $S_0$ denotes the current state of the system; $\ldots S_2 S_1$ denotes the history of the system; and $S_1 S_2 \ldots$ denotes the future of the system. Further, let $\sigma_j$ denote the sequence of states $\ldots S_j S_{j-1}$ or the sequence of states $S_j S_{j+1} \ldots$, depending on $j < 0$ or $j > 0$. Further, we use $\delta(<\text{op}>, S_j)$ to denote the state resulting from executing the operation $<\text{op}>$ in state $S_j$.

The semantics of the temporal assertions are as follows.

- $\sigma \wedge w$ iff $S_0 \models w$ where $w$ is not a temporal assertion, i.e., $w$ is a first order wff whose free variables are parameters.

- $\sigma \text{always} w$ iff $(\forall j < 0)(\sigma_j \models w)$, where $w$ is any temporal assertion. (i.e., always in the past excluding the present $w$ is true.)

- $\sigma \text{always}^+ w$ iff $(\forall j < 0)(\sigma_j \models w)$, where $w$ is any temporal assertion. (i.e., always in the past including the present $w$ is true.)

- $\sigma \text{ever} w$ iff $(\exists j < 0)(\sigma_j \models w)$, where $w$ is any temporal assertion. (i.e., ever in the past excluding the present $w$ is true.)

- $\sigma \text{ever}^+ w$ iff $(\exists j < 0)(\sigma_j \models w)$, where $w$ is any temporal assertion. (i.e., ever in the past including the present $w$ is true.)

- $\sigma \text{always} w$ iff $(\forall j > 0)(\sigma_j \models w)$, where $w$ is any temporal assertion. (i.e., always in the future excluding the present $w$ is true.)
- \(\sigma^{\text{always}} w\) iff \(\forall j > 0\)(\(\sigma_j \mathcal{H} w\)), where \(w\) is any temporal assertion. (i.e., always in the future including the present \(w\) is true.)

- \(\sigma^{\text{ever}} w\) iff \(\exists j > 0\)(\(\sigma_j \mathcal{H} w\)), where \(w\) is any temporal assertion. (i.e., ever in the future excluding the present.)

- \(\sigma^{\text{ever}+} w\) iff \(\exists j > 0\)(\(\sigma_j \mathcal{H} w\)), where \(w\) is any temporal assertion. (i.e., ever in the future including the present \(w\) is true.)

- \(\sigma \not\mathcal{H} w\) iff \(\sigma \not\mathcal{H} w\), where \(w\) is a temporal assertion.

- \(\mathcal{P} \text{EXECUTABLE}(\langle \text{op} \rangle)\) iff the following conditions hold:
  1) \(\sigma\) the temporal assertion of the operation \(\langle \text{op} \rangle\);
  2) \(S_0 \models \text{the precondition of the operation } \langle \text{op} \rangle\); and
  3) \(\delta(\langle \text{op} \rangle, S_0) \models \text{the postcondition of the operation } \langle \text{op} \rangle\).

- \(\sigma \mathcal{H} (w_1 \& w_2)\) iff \(\sigma \mathcal{H} w_1\) and \(\sigma \mathcal{H} w_2\), where \(w_1, w_2\) are temporal assertions.

- \(\sigma \mathcal{H} (w_1 \lor w_2)\) iff \(\sigma \mathcal{H} w_1\) or \(\sigma \mathcal{H} w_2\), where \(w_1, w_2\) are temporal assertions.

- \(\sigma \mathcal{H} (w_1 \rightarrow w_2)\) iff either not \(\sigma \mathcal{H} w_1\) or \(\sigma \mathcal{H} w_2\), where \(w_1, w_2\) are temporal assertions.

We must define the semantics for \(\sigma_j\), recalling that \(\sigma_j = \ldots S_{j-1} S_j\) when \(j < 0\) or \(\sigma_j = S_j S_{j+1} \ldots\) when \(j > 0\):

- \(\sigma_j \mathcal{H} w\) iff \(S_j \mathcal{H} w\) where \(w\) is not a temporal assertion.

- \(\sigma_j^{\text{always}} w\) iff \(j < 0\) and \(\forall k < j\)\(\sigma_k \mathcal{H} w\) where \(w\) is any temporal assertion.

- \(\sigma_j^{\text{ever}} w\) iff \(j < 0\) and \(\forall k < j\)\(\sigma_k \mathcal{H} w\) where \(w\) is any temporal assertion.

- \(\sigma_j^{\text{ever}+} w\) iff \(j > 0\) and \(\forall k > j\)\(\sigma_k \mathcal{H} w\) where \(w\) is any temporal assertion.

- \(\sigma_j^{\text{ever}+} w\) iff \(j < 0\) and \(\exists k < j\)\(\sigma_k \mathcal{H} w\) where \(w\) is any temporal assertion.

- \(\sigma_j^{\text{ever}+} w\) iff \(j > 0\) and \(\exists k > j\)\(\sigma_k \mathcal{H} w\) where \(w\) is any temporal assertion.
- $\sigma_j \models_{ever} \cdot w$ iff $j \geq 0$ and $(\exists k \geq j)(\sigma_k \models w)$ where $w$ is any temporal assertion.

- $\sigma_j \models_{\neg} w$ iff not $\sigma_j \models w$, where $w$ is any temporal assertion.

- $\sigma_j \models_{\text{EXECUTABLE}}(\langle \text{op} \rangle)$ iff the following conditions hold:
  1) $\sigma_j \models$ the temporal assertion of the operation $\langle \text{op} \rangle$;
  2) $S_j \models$ the precondition of the operation $\langle \text{op} \rangle$; and
  3) $\delta(\langle \text{op} \rangle, S_j) \models$ the postcondition of the operation $\langle \text{op} \rangle$.

- $\sigma_j \models (w_1 \ & \ w_2)$ iff
  $\sigma_j \models w_1$ and $\sigma_j \models w_2$.
  where $w_1$ and $w_2$ are temporal assertions.

- The definition is similarly extended to include $\lor$ and $\rightarrow$.

### 3.2.4 Examples

As an example, the temporal constraint stating that "Whoever has been an employee (probably has been fired) cannot be hired again." can be expressed as follows:

$\models_{ever} \cdot E(x) \rightarrow \models_{\text{EXECUTABLE}}(\text{hire}(x))$

The interpretation of this formula is that if sometimes in the past or at the present $x$ is an employee, then the hire($x$) operation is not executable. Clearly, $x$ here should be understood as an arbitrary individual.

The temporal constraint stating that an employee's salary must never decrease can be expressed as:

$\models_{\text{always}} \cdot (\forall z)(\models \text{EARN}(\bar{x}, z) \rightarrow z \geq \bar{y})$

The interpretation of this formula is that if $\bar{x}$ earns $\bar{y}$ at the present, then always in the future for all $z$, $\bar{x}$ earns $z$ implies $z$ is greater than or equal to $\bar{y}$.

### 3.3 Operation Description

In Sect.2.5.3, we pointed out that the description of an operation should include information referring to the history of the system. This information prevents the execution of illegal sequences of operations. Thus, an operation description consists of three parts: a temporal assertion which specifies the desirable system history under which the operation can be performed; a precondition of the operation which specifies the condition that must be satisfied by a state in which the operation is going to be executed; a postcondition of the operation which specifies the "result" of executing the operation.
3.3.1 The Formal Definition of an Operation

Syntactically, an operation is defined (in Backus Normal Form) as follows:

\[
\begin{align*}
\text{operation} & ::= \text{op}: \text{op-desc} \\
\text{op} & ::= \text{op-name}(\langle \text{parameter-list} \rangle) | \text{op-name}^* (\langle \text{parameter-list} \rangle) \\
\text{op-name} & ::= \text{string of lower-case letters} \\
\text{op-desc} & ::= \langle \text{tap} \rangle, S_i \models \langle \text{pre} \rangle \implies \delta(\langle \text{op} \rangle, S_i) \models \langle \text{post} \rangle \\
\text{tap} & ::= \text{any} | \langle \text{tap} \rangle \\
\text{pre} & ::= \langle \text{wff} \rangle \\
\text{post} & ::= \langle \text{wff} \rangle
\end{align*}
\]

In the above definition, \( \langle \text{tap} \rangle, \langle \text{parameter-list} \rangle, \) and \( \langle \text{wff} \rangle \) are as defined in Sect.3.2.1. any is used as a dummy temporal assertion when there is no need to refer to the past. By convention, any is implied by every temporal assertion (see Sect.4.10.2.1).

In the above definition, \( \delta(\langle \text{op} \rangle, S_i) \) is a mapping from a pair of one operation and a state to a new state. This notation is influenced by [501]. A similar mapping is also defined in [35]. An intuitive interpretation of \( \delta(\langle \text{op} \rangle, S_i) \) is that in state \( S_i \), if one executes the operation denoted by \( \langle \text{op} \rangle \), then the system enters a new state designated by \( \delta(\langle \text{op} \rangle, S_i) \). In other words, \( \delta(\langle \text{op} \rangle, S_i) \) denotes the new state resulting from executing operation \( \langle \text{op} \rangle \) in state \( S_i \).

3.3.2 Some Examples

The following examples serve to illustrate the specification of some operations. The meanings of the predicate symbols are as defined in Sect.3.1.6. The intention of the operations can be understood from the strings of letters that define their names.

\[\text{hire}(\tilde{x}): \text{always}^* \text{E}(\tilde{x}), S_i \models \text{E}(\tilde{x}) \implies \delta(\text{hire}(\tilde{x}), S_i) \models \text{E}(\tilde{x})\]

This expression says that if it was always true in the past and it is true in the state \( S_i \) that \( \tilde{x} \) is an employee, then in the state resulting from hiring \( \tilde{x} \), it is true that \( \tilde{x} \) is an employee.

\[\text{fire}(\bar{x}): \text{any}, S_i \models \text{E}(\bar{x}) \text{M}(\bar{x}) \implies \delta(\text{fire}(\bar{x}), S_i) \models \text{E}(\bar{x})\]

This expression says that if it is true in state \( S_i \) that \( \bar{x} \) is an employee but not a manager, then in the state resulting from firing \( \bar{x} \), it is true that \( \bar{x} \) is not an employee.

\[\text{raise}^*(\tilde{x}, 10\% \bar{y}): \text{any}, S_i \models \text{E}(\tilde{x}) \text{EARN}(\tilde{x}, \bar{y}) \implies \delta(\text{raise}^*(\tilde{x}, 10\% \bar{y}), S_i) \models \text{EARN}(\tilde{x}, \bar{y}+10\% \bar{y})\]

This expression says that if it is true in state \( S_i \) that \( \tilde{x} \) is an employee and \( \tilde{x} \) earns \( \bar{y} \), then in the state resulting from raising the salary of \( \tilde{x} \) by \( 10\% \), it is true that \( \tilde{x} \) earns \( \bar{y}+10\% \bar{y} \).

\[\text{promote}(\bar{x}): \text{any}, S_i \models \text{E}(\bar{x}) \text{M}(\bar{x}) \implies \delta(\text{promote}(\bar{x}), S_i) \models \text{M}(\bar{x})\]

This expression says that if it is true in state \( S_i \) that \( \bar{x} \) is an employee but not a manager, then in the state resulting from promoting \( \bar{x} \), it is true that \( \bar{x} \) is a manager.
\text{demote}(\bar{x}) : \exists y, S_i \models E(\bar{x}) \land M(\bar{x}) \Rightarrow \delta(\text{demote}(\bar{x}), S_i) \models \neg M(\bar{x})

This expression says that if it is true in state $S_i$ that $\bar{x}$ is an employee and $\bar{x}$ is a manager, then in the state resulting from demoting $\bar{x}$, it is true that $\bar{x}$ is not a manager.

These examples show how we use the semantical notation \text{"\models\"} to relate a system state and a condition which is expected to hold in the state.

3.3.3 The Formal Semantics

An information system is an evolutionary object. The evolution of the information system is characterized by the execution of the operations. Each execution of a sequence of operations results in a history of the information system. As shown in Fig.3, such a history can be represented by a sequence of legal system states $\ldots S_{i-1}S_i$ with $S_i$ being considered as the current state. For all $j < i$, state $S_{j+1}$ is the result of executing some operation in state $S_j$. Thus, by definition, an operation $\langle op \rangle$ is allowed to be executed in a state $S_j$ iff

1) The history of the system satisfies the temporal assertion of the operation. That is, $\ldots S_{i-1}S_i$ satisfies the temporal assertion of the operation.

2) The current state $S_i$ satisfies the precondition of the operation. That is, the applicability condition of the operation in question [35].

3) There is a legal system state $\delta(\langle op \rangle, S_i)$ which satisfies the postcondition of the operation. The postcondition asserts something which probably did not hold in state $S_i$. To ensure that the new system state will not violate the static constraints, we need to find out whether there is a legal system state in which the postcondition is true. If there is a legal system state (probably more than one), say $S_k$, which satisfies the postcondition of the operation, then we could take $S_k$ as the state resulting from applying the operation $\langle op \rangle$ in state $S_i$, i.e., $S_k = \delta(\langle op \rangle, S_i)$. In this case, we say that the application of the operation yields some legal system state $S_k$ [35]. We call this test acceptability analysis. Since there could be more than one legal system state $S_k$ which satisfies the postcondition of the operation, there could be more than one transition from state $S_i$. However, not all of these transitions are the desirable transitions (conf. Sect.2.5.2).

4) Therefore, we also have to investigate if the transition from state $S_i$ to state $S_k$ is the desirable transition. The reason is that the execution of an operation changes only a few assertions. Most of the assertions that held in the "old" state $S_i$ remain held in the "new" state $\delta(\langle op \rangle, S_i)$. Thus, the desirable transition from $S_i$ to $S_k$ must preserve all the assertions that should remain hold in the new state. This test is called the preservability analysis. It can also be seen as a test for finding undesirable side-effects.

Clearly, if the above 4 criteria are satisfied by the execution of an operation, then the system will enter the state $S_k$ as described above. Moreover, a new history of the system will be recorded. The problem
that has not been discussed above is how to define the preservability criterion. This can be done by comparing the logical consequences of the postcondition and the logical consequences of the precondition. This is to be discussed in the next chapter.

Now how can we check if the system’s history satisfies the temporal assertion of an operation? Let \( \sigma_i = \ldots S_{i-1} S_i \) denote a sequence of legal system states. Let \( w \) denote the temporal assertion of the operation. Then the notion that \( \sigma_i \) satisfies \( w \), denoted as \( \sigma_i \models w \), is defined as follows.

- \( \sigma_i \models w \) iff \( S_i \models w \), where \( w \) is not a temporal assertion.
- \( \sigma_i \models\text{always}^+ w \) iff \( \forall j \langle i \rangle (\sigma_j \models w) \)
- \( \sigma_i \models\text{ever}^+ w \) iff \( \exists j \langle i \rangle (\sigma_j \models w) \)
- \( \sigma_i \models\text{fan} \)

Let \( w_1 \) and \( w_2 \) denote two temporal assertions to the past excluding the present as defined in Sect.3.2.3. Then

- \( \sigma_i \models w_1\& w_2 \) iff \( \sigma_i \models w_1 \) and \( \sigma_i \models w_2 \)
- \( \sigma_i \models w_1\vee w_2 \) iff \( \sigma_i \models w_1 \) or \( \sigma_i \models w_2 \)
- \( \sigma_i \models \neg w \) iff not \( \sigma_i \models w \)
- \( \sigma_i \models w_1 \rightarrow w_2 \) iff either not \( \sigma_i \models w_1 \) or \( \sigma_i \models w_2 \)

3.3.4 Related Work in Artificial Intelligence

Problem-solving is one of the branches of artificial intelligence. Problem-solving methods solve problems by searching for a solution in a space of possible solutions. In particular, some problem-solving methods start from an initial configuration (or initial state) and try to reach a goal configuration (or goal state) by a sequence of applications of some problem-defined operators.

There are two main items in the definition of each operator:

- An applicability condition which is a wff, stating the condition under which the operator is applicable. This corresponds to the precondition in our framework.
- Rules for transforming the set of wffs describing the state to which the operator is applied to a new set of wffs describing the resulting state.

The transformation rules take the form of a list of wffs to be deleted and a list to be added, assuming that those wffs not deleted remain in the new state description. The added statements together with the deleted statements correspond to the postcondition in our framework. Indeed, an added statement is specified as a false/true transition, and a deleted statement is specified as a true/false transition in the present framework. A similar treatment is found in [84].
3.3.4.1 The Monkey and Bananas Example

The monkey-and-bananas example is often used in artificial intelligence to demonstrate problem-solving methods [72]. The problem can be stated as: A monkey is in a room containing a box and a bunch of bananas. The bananas are hanging from the ceiling out of reach of the monkey. What sequence of actions will allow the monkey to obtain the bananas? One sequence of actions can be that the monkey goes to the box, pushes it under the bananas, climbs on top of the box and grasps the bananas.

In [72], it is shown that a "state" of the monkey-and-bananas problem can be defined by a set of first order wffs. The actions of the monkey can be specified by a set of operators each of which changes one set of wffs into another set of wffs. The set of goal states can be defined as being described by any set of wffs from which some goal wff follows.

For example, the initial state $S_0$ of the monkey-and-bananas problem can be defined by the following wffs:

- ¬ONBOX: it is true when the monkey is not on the box.
- AT(box,b): the box is at position b.
- AT(monkey,a): the monkey is at position a.
- ¬HB: it is true when the monkey does not have the bananas.

A set of actions (or operators) can also be defined for the monkey. The actions include goto(u), pushbox(v), climbbox, and grasp, where u, v are variables. For instance, the action goto(u) is defined as:

- Applicability condition: ¬ONBOX (i.e., the precondition)
- Transformation rule (i.e., the postcondition):
  - Delete: AT(monkey,$)
  - Add: AT(monkey,u)

where the $ sign stands for any term. The wff AT(monkey,$) is to be deleted regardless of the value of $. Since u is a variable, goto(u) should be understood as goto anywhere. The application of the operation will result in a state-description schema which contains u as a variable. Assigning some particular constant to u will produce a particular state description.

Since the applicability condition of goto(u), i.e., ¬ONBOX, is true in the initial state, therefore we can apply goto(u) in the initial state. The resulting state is:

- ¬ONBOX
- AT(box,b)
- AT(monkey,u)
We see that only one wff, i.e., AT(monkey, a) has been changed to AT(monkey, u). The other wffs remain in the new state description. The problem posed by the evident fact that operators affect certain wffs but do not affect the others is called the "frame problem" by McCarthy [65].

An improved representation of the monkey-and-bananas problem is also shown in [72], which is based upon the work of Green [40] [41] [42]. The improvement is to include a state term in each of the predicates.

In [50], both states and statements are regarded as individuals and are represented by means of terms. Thus, the term

\[ \text{result}(\text{goto}(u), S_0) \]

may be used to represent the state which results from applying the action goto(u) in state \( S_0 \). The notion that a statement \( x \) is true in a state \( y \) is represented by the binary relation

\[ \text{HOLD}(x, y) \]

For instance, the initial state \( S_0 \) can be defined by the following assertions

- \( \text{HOLD}(\text{"ONBOX}, S_0) \)
- \( \text{HOLD}(\text{AT(box,b)}, S_0) \)
- \( \text{HOLD}(\text{AT(monkey,a)}, S_0) \)
- \( \text{HOLD}(\text{"HB}, S_0) \)

The operator goto(u) can be defined by the following wffs:

- Added statements
  - \( \text{HOLD}(\text{AT(monkey}, u), \text{result}(\text{goto}(u), s)) \), which is intended to express that in any state \( s \), the application of the operator goto(u) results in a state in which AT(monkey, u) holds.

- Frame axiom and deleted statements

\[ \text{HOLD}(v, s) \& \text{DIFF}(v, \text{AT(monkey}, s)) \rightarrow \text{HOLD}(v, \text{result}(\text{goto}(u), s)) \]

which expresses that if in an arbitrary state \( s \), statement \( v \) holds and \( v \) is different from AT(monkey, s), then \( v \) remains true in the state which results from applying goto(u) in s.

The predicate DIFF(\( t_1, t_2 \)) holds for every variable-free terms \( t_1 \) and \( t_2 \) when \( t_1 \) and \( t_2 \) are syntactically distinct. In other words, \( t_1 \) and \( t_2 \) do not match. Note that the definition of the goto(u) operator should be supplemented by infinitely many clauses of the form
\[ \text{DIFF}(t_i, t_j) \]

for every pair of terms \( t_i, t_j \) which do not match.

As pointed out earlier, our framework for specifying the operations is partly influenced by the work of Kowalski. That is, we make use of the mapping

\[ \delta: \{ \text{operation} \} \times \{ \text{state} \} \longrightarrow \{ \text{state} \} \]

which is influenced by [50]. The difference is that in our framework, the semantical notion "\( \vdash \)" is used. Moreover, the precondition and postcondition are clear from the syntax of the operation specification. Another important difference is that in our framework, a historic condition for applying the operation is explicitly specified. To the opinion of this author, the operation description that is proposed in this framework is more understandable.

3.3.4.2 Comparative Remarks

Several remarks concerning the differences between the framework used in the problem-solving field and in this thesis are:

- In the problem-solving field, the objective is to represent the problem in a "clever" way so that the search for an "optimal" solution from a possible solution space can be as efficient as possible. Therefore, problem representation and intelligent search strategy can be considered as the two main tasks in the problem-solving field. On the other hand, the specification of an information system pays more attention on the understandability of the specification.

- In the problem-solving field, the formulation of a problem-solving schema usually must also take into account the efficiency of the search strategy. The reason is that the formulation of the schema is a part of the solution of a problem. The schema is going to be executed. As a result, which statements are to be added and which statements are to be deleted must be listed explicitly. Without considering in detail what to do and how to do it, the problem can never be solved.

On the other hand, the specification of an information system should be as free as possible from data processing considerations [20] [84] [51]. The efficiency consideration should be postponed until the detailed design stage only then a cost-benefit decision becomes possible. Thus, an imperative style of specification should be avoided in the specification field [102]. As a consequence, only the result or the behavior of the "computation" is declaratively specified without specifying in full detail the mechanisms which will carry out the computation.

The specification of the precondition and the postcondition of the operation seems to be one of the approaches that is used in the literature. It seems unnecessary to explicitly
formulate the "frame axioms". The effect of the operation is specified by a pair of one precondition and one postcondition, anything that is not affected by the operation remains unchanged. It is also impossible in practice to explicitly formulate the "frame axioms" because an information system usually involves a large number of "relations" and operations. Explicitly formulating the "frame axioms" implies that a frame axiom should be specified for each operation and a long list of DIFF(x,y) assertions for each pair x,y of "relations". Since an information system specification is not meant to be an executable object, the formulation of the frame axioms and the list of DIFF(x,y) seems to be redundant.

- In the problem-solving field, it is often possible to describe an initial state and a goal state. However, in the information system field, there is no goal state. It is usually not easy to identify an initial state. To the opinion of this author, we have instead legal states and the notion of current state [35] [22] [84] [80]. Note that an empty state (e.g., there is no tuple in any relation of a data base) is not necessarily an "initial state". As discussed somewhere earlier, a necessary condition for considering an empty state as an initial state is that the empty state is a legal state. That is, the empty state should satisfy all the static constraints.

- The transformation from one state to another state in the problem-solving field has been purposely controlled by some "artificial intelligence" (or "built-in intelligence") so that the expected goal state can be reached as soon as possible. However, in the information system field, the evolution of the system depends very much on external events and state changes [10] [80] [20] [84]. Thus, clear and concise description of the desirable state transitions rather than intelligent formulation of the problem solving strategy is important in specifying an information system.

It is beyond the scope of this thesis to treat the difference between the two fields in more detail.
CHAPTER 4:

CONSISTENCY CHECKING:

THE MODIFIED TABLEAUX APPROACH
4 CONSISTENCY CHECKING: THE MODIFIED TABLEAUX APPROACH

4.1 Syntactical and Semantical Approaches

One of the advantages of using first order logic for specification is that it has powerful and complete theorem proving procedures for deductive reasoning. The proof procedures in first order logic can be classified into two main classes:

- The syntactical approach; and
- The semantical approach.

The syntactical approach usually reasons upon the syntax of the language using a set of inference rules. The process of reasoning can be seen as substituting a string of symbols for another string of symbols. That is, the conclusion that is drawn is independent of the underlying "meaning" of the symbols of the language. Within the framework of the first order language, a formula or a sentence can be seen as a string of symbols while a set of formulae can be seen as a set of strings. Thus, a formula or a sentence is deducible from a set of formulae iff it can be obtained from the set of formulae through a sequence of symbolic substitutions by using the inference rules.

It is well-known that syntactical approaches are effective in proving that a set W of wffs is inconsistent, see e.g., [9] [33] [72] [75] [78] ect. The idea behind the proof of the inconsistency is to direct the proof procedure toward the deduction of a wff w as well as its negation ~w from W. This implies that any model M which satisfies W must satisfy w. Since ~w is also a consequence of W, which implies that M must satisfy w as well. However, it is impossible to have a model M which satisfies w and ~w at the same time and hence M cannot be a model of W. That is, W has no model. Thus, by the extended completeness theorem, W is inconsistent.

The semantical approach is based upon the concept of interpretation (See Sect.3.1.3). Simply speaking, an interpretation is a mapping between the symbols of the language and some "possible world". That is, to each non-logic symbol of the language, we associate a thing in the possible world. This correspondence is intended to give some "meaning" to the symbols of the language. Thus, the consistency of a set W of wffs can be concluded iff a model (in the sense of Sect.3.1.4) for W can be constructed. The construction of the model tries to interpret the symbols appearing in the wffs in such a way that the model which is constructed will satisfy all the wffs. If such an attempt succeeds, then the set W of wffs is consistent. We see that rather than reasoning upon the syntax of the language, this approach reasons upon the semantics of the language.

Note also that instead of proving the inconsistency of a set W of wffs, a semantic approach tries to prove that W is consistent by constructing a model for it [23].

The first order tableau system that is to be presented in Sect.4.4 belongs to the syntactical approach. The modified tableau approach belongs to the mixture of syntactical and semantical approaches: the
construction of the tableaux tree is syntactical while the construction of the system states is semantical (Sect. 4.7 and Sect. 4.8).

4.2 An Overview of the Modified Tableaux Approach

In this thesis, we present two approaches for testing the consistency of a conceptual model which is specified by using the framework of the last chapter:

- The first approach uses a modified first order tableaux system, which is to be presented in this chapter.

- The second approach uses a modified resolution method, which is to be presented in the next chapter.

The reason for presenting the modified tableaux approach is two folds:

1) We use it to clarify some of the ideas in formal consistency checking, since the modified tableaux approach is more straightforward and hence easier to be understood.

2) We use it as a comparison for the second approach.

4.2.1 Verification of Static Constraints

In the modified tableaux approach, we assume that the static constraints are specified in a many-sorted logic. Further, we do not allow any recursiveness among the (sorted) universal and existential quantifiers. (A similar restriction is found in [67] where some halting conditions for terminating infinite derivation paths in a deductive database are defined.) By this restriction, the modified tableaux approach can always determine the consistency as well as the inconsistency of the static constraints. If the static constraints are not consistent, then the approach will sooner or later detect the inconsistency. On the other hand, if the static constraints are consistent, then the approach will construct a set of legal system states each of which satisfies all the static constraints.

4.2.2 Operation Analysis

Using these system states we then analyse the operation specifications. For each of the operation descriptions, we analyse if there is a legal system state in which the precondition of the operation holds. If so, then the operation is said to be applicable in the legal system state. The legal system state is called the prestate of the operation. We then analyse if there is a legal system state in which the postcondition of the operation holds. If so, then the operation is said to be accepted by the legal system state, which is called the poststate of the operation. Finally, we analyse if the transition from the prestate to the poststate preserves those assertions that are expected to remain true in the poststate. The analysis of all the operations in this way will result in a state transition diagram. Note that from a state transition point of view, we may temporarily ignore the temporal assertions of the operation.
descriptions. The temporal assertions of the operation descriptions will be taken into account when we test the consistency of the temporal constraints.

4.2.3 Verification of Temporal Constraints

The consistency checking of the temporal constraints is carried out as follows.

1) First we transform the state transition diagram into a family of finite automata $f_i$ each of which has one of the state of the transition diagram as its initial state. For each $f_i$, we construct a pushdown automaton $p_i$ by taking into account the temporal assertions of the operations.

2) Secondly, we generate a set $T_i$ of test sequences for each $f_i$, obtained in the last step. A test sequence is a sequence of operations that can possibly be performed in the future information system.

3) Thirdly we use the corresponding pushdown automaton $p_i$ to analyse the acceptance of the test sequences in $T_i$. Intuitively, a sequence of operations in $T_i$ can be executed in the future information system if it is accepted by the pushdown automaton $p_i$.

4) Finally, the consistency of the temporal constraints is proved if each test sequence that is accepted by some pushdown automaton $p_i$ satisfies all the temporal constraints.

4.2.4 The Layout of the Chapter

We begin with Sect. 4.3 which discusses the unsolvability of the first order logic. This implies that certain restriction must be imposed upon the static constraints or we cannot effectively determine the consistency of the static constraints. Sect. 4.4 presents the first order tableaux system. Sect. 4.5 discusses the restriction that is imposed. Sect. 4.6 describes the method for detecting the violation of the restriction. Sect. 4.7 presents the modified tableaux approach. Sect. 4.8 presents the method of constructing the system states. Sect. 4.9 analyses the operation descriptions and finally Sect. 4.10 describes the consistency checking of the temporal constraints.

4.3 The Un solvability of First Order Logic

By unsolvability we mean that there in general does not exist any effective algorithm which, given an arbitrary set of first order logic formulae, can determine in a finite number of steps, whether the set of formulae is consistent [72] [60].

According to [60], the following 9 classes of prenex formulae containing only unary and binary predicate symbols are not solvable:
| 1) \(\forall \forall (n, 1)\) | 2) \(\forall \exists (0, 1)\) | 3) \(\forall \exists (0, 1)\) | 4) \(\exists \forall \forall \exists (0, 1)\) | 5) \(\forall \forall \exists (0, 1)\) | 6) \(\forall \forall \forall (n, 1)\) | 7) \(\exists \forall \forall (0, 1)\) | 8) \(\exists \forall (0, 1)\) | 9) \(\forall \exists (0, 1)\) |

where \(\forall\) and \(\exists\) represent a universal and an existential quantifier respectively. A '\(*\)' which immediately follows a quantifier indicates that there is zero or more consecutive appearances of the quantifier. The first and the second number enclosed in the parentheses is the number of unary and binary predicate symbols used in the class of formulae. For instance, \(\forall \exists (0, 1)\) means that the class of formulae which are defined by zero unary predicate symbol and one binary predicate symbol, and which have the form that there are zero or more universal quantifiers followed by an existential quantifier, is unsolvable. That is, there exists no effective method which can prove the consistency of a set of formulae falling into this class.

Note that the above result does not imply that the inconsistency of a set of formulae falling into anyone of the 9 classes is undecidable. On the contrary, the inconsistency of a set of first order logic formulae has been proved to be decidable [60] [9]. In this sense, we say that the first order logic is semidecidable.

Research in the field of computability indicates that although first order logic is in general unsolvable, however certain sub-classes of first order formulae are decidable [59] [60]. Therefore, in order to determine the consistency of a set of wffs, we must impose some restrictions upon the form of the formulae.

### 4.4 The First Order Tableaux System

The first order tableaux system is a deduction system by which one can determine whether a set of wffs is inconsistent. Given a set \(W\) of wffs, the tableaux system tries to construct a confutation for the set \(W\) of wffs. A confutation of a set \(W\) of wffs is a proof that \(W\) is inconsistent.

#### 4.4.1 The Deduction Rules

The tableaux system uses a set of deduction rules, each of which is a theorem or an inference rule of the first order logic. Some of the deduction rules are listed on the next page. The corresponding first order theorems or rules are listed in the "comment" column.
<table>
<thead>
<tr>
<th>rule name</th>
<th>rule</th>
<th>comment</th>
</tr>
</thead>
</table>
| ↔-rule | ¬¬w | ¬¬w (≡) w  
i.e., from ¬¬w,  
we may infer w. |
| &-rule | ¬(w1&w2) | ¬(w1&w2) (≡) ¬w1 ∨ ¬w2  
i.e., if ¬(w1&w2) is true, then ¬w1 must be true or ¬w2 must be true. |
| V-rule | ¬(w1Vw2) | ¬(w1Vw2) (≡) ¬w1 & ¬w2  
i.e., if ¬(w1Vw2) is true, then both ¬w1 and ¬w2 must be true. |
| →-rule | ¬(w1 → w2) | ¬(w1 → w2) (≡) w1 & ¬w2  
i.e., from ¬(w1 → w2) infer w1 and ¬w2 |
| &-rule | w1 & w2 | If w1 & w2 is true, then both w1 and w2 are true. |
| V-rule | w1 V w2 | If w1 V w2 is true, then either w1 or w2 must be true. |
| →-rule | w1 → w2 | w1 → w2 (≡) ¬w1 V w2  
i.e., w1 → w2 iff either ¬w1 or w2. |

Fig. 4a. The propositional deduction rules
<table>
<thead>
<tr>
<th>rule name</th>
<th>rule</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>(∀-rule)</td>
<td>(∀x)w(x) : _ _ w(y) _ _</td>
<td>(∀x)w(x) \iff w(y) for any y.</td>
</tr>
<tr>
<td>(¬∀-rule)</td>
<td>(¬(∀x)w(x) : _ _ ¬w(y) _ _</td>
<td>It is not the case that for every x, w(x) is true, then it is not the case that w(y) is true for some y. That is, y must be new to the proof.</td>
</tr>
<tr>
<td>(∃-rule)</td>
<td>(∃x)w(x) : _ _ w(y) _ _</td>
<td>y is new.</td>
</tr>
<tr>
<td>(¬∃-rule)</td>
<td>(¬(∃x)w(x) : _ _ ¬w(y) _ _</td>
<td>There exists an x such that w(x) is true implies that w(y) is true for y being new to the proof.</td>
</tr>
</tbody>
</table>

*Fig. 4b. The quantificational deduction rules*

Three deduction rules corresponding to the identity axioms defined in Sect. 3.1.2 are listed below. These three rules can be applied at any time:

a) The Self-Identity rule (SI-rule) which corresponds to the reflexive axiom u=u:

\[
\begin{align*}
\vdash & \\
\_ \_ & t = t \\
\text{for any term } t
\end{align*}
\]

b) The Substitution in Functions rule (SF-rule) which says that two instances of a function are identical if their respective arguments are identical:

\[
\begin{align*}
\vdash & \\
\_ \_ & t_{i1}=t_{j1} \ldots t_{im}=t_{jm} \implies f(t_{i1}, \ldots, t_{im})=f(t_{j1}, \ldots, t_{jm})
\end{align*}
\]
c) The Substitution in Predicate rule (SP-rule) which says that an instance of a predicate implies another instance of the predicate if their respective arguments are identical:

\[ t_{i_1} = t_{j_1} \rightarrow \ldots \rightarrow t_{i_m} = t_{j_m} \rightarrow P(t_{i_1}, \ldots, t_{i_m}) \rightarrow P(t_{j_1}, \ldots, t_{j_m}) \]

### 4.4.2 The Tableaux Method

The confutation of a set \( W \) of wffs can be achieved by constructing a tableau tree for \( W \). In order to facilitate the presentation of the construction of a tableau tree, we first define what is meant by a ground instance of a predicate. Let

\[ P(x_1, \ldots, x_n) \]

be an \( n \)-ary predicate symbol, where \( x_i \) are variables for \( i = 1, \ldots, n \). A ground instance of \( P(x_1, \ldots, x_n) \) is of the form

\[ P(t_1, \ldots, t_n) \]

where for \( i = 1, \ldots, n \), \( t_i \) is a term with no variables.

Let \( W \) be a set of wffs which is to be confuted. The construction of the tableau tree starts with \( W \) as the root of the tree which is defined as the initial tableau tree, \( T_0 \).

Suppose that \( T_i \) has been constructed, then \( T_{i+1} \) is obtained by picking up, in a branch of \( T_i \), one of the formulae for which one of the rules can be applied; extend the branch of \( T_i \) by the result of applying the rule. That is, we extend the branch to one or two longer branches depending on the rule which is used. A branch is immediately closed if it is found to contain a ground instance of a predicate and the negation of that ground instance. A closed branch will not be expanded further. A tableau is called a closed tableau if all its branches are closed. The process is repeated until a closed tableau is constructed. In this case we say that the set of formulae denoted by the root of the closed tableau is inconsistent. The process of constructing a closed tableau is called a confutation for the set of formulae represented by the root.

The above presentation is adapted from [9].

### 4.4.3 An Example

As an example, we show the inconsistency of the following set of wffs, where \( WF(x,y) \) is informally agreed to represent that \( x \) works for \( y \).

1) John works for any one who works for the top-manager.
\[
(\forall x)( WF(x, 'top-manager') \rightarrow WF(John, x) )
\]

2) No one works for anyone who works for himself.
\[
(\exists x)(\exists y)( WF(x, x) \& WF(y, x) )
\]
3) John works for the top-manager.
    WF(John, 'top-manager')

The closed tableau is constructed as depicted in Fig.5a through
Fig.5.e. For clarity, we repeat each step together with the rule that
is applied. Note that the wffs in the figures have been labeled by
numbers.

STEP 1. Apply V-rule to 1) with John replacing x results in 4).
1)  (\forall x)[ WF(x, 'top-manager') \rightarrow WF(John, x) 
2)  \neg(\exists x)(\exists y)( WF(x, x) \& WF(y, x) ) 
3)  WF(John, 'top-manager') 
    \rightarrow
4)  WF(John, 'top-manager') \rightarrow WF(John, John)

Fig.5a. The first step of the construction

STEP 2. Apply \rightarrow rule to 4) results in 5) and 6). Note that 5)
conflicts with 3), so we immediately closed the branch.
1)  (\forall x)[ WF(x, 'top-manager') \rightarrow WF(John, x) 
2)  \neg(\exists x)(\exists y)( WF(x, x) \& WF(y, x) ) 
3)  WF(John, 'top-manager') 
    \rightarrow
4)  WF(John, 'top-manager') \rightarrow WF(John, John) 
    \rightarrow
5)  \neg WF(John, 'top-manager')  6)  WF(John, John) 
    X conflict with 3)

Fig.5b. The second step of the construction

STEP 3. Apply \neg I-rule to 2) with John replacing x results in 7).
1)  (\forall x)[ WF(x, 'top-manager') \rightarrow WF(John, x) 
2)  \neg(\exists x)(\exists y)( WF(x, x) \& WF(y, x) ) 
3)  WF(John, 'top-manager') 
    \rightarrow
4)  WF(John, 'top-manager') \rightarrow WF(John, John) 
    \rightarrow
5)  \neg WF(John, 'top-manager')  6)  WF(John, John) 
    X conflict with 3)
7)  \neg(\exists y)(WF(John, John)\&WF(y, John))

Fig.5c. The third step of the construction
STEP 4. Apply "3-rule to 7) with John replacing y results in 8).
1)  $(\forall x)(\text{WF}(x, 'top-manager')) \rightarrow \text{WF}(\text{John}, x)$
2)  $\neg [\exists x](\exists y)(\text{WF}(x, x) \& \text{WF}(y, x))$
3)  $\text{WF}(\text{John}, 'top-manager')$

4)  $\text{WF}(\text{John}, 'top-manager') \rightarrow \text{WF}(\text{John}, \text{John})$

5)  "WF(John, 'top-manager')" 6) WF(John, John)
X conflict with 3)

7)  $\neg [\exists y](\text{WF}(\text{John}, \text{John}) \& \text{WF}(y, \text{John}))$

8)  $\neg (\text{WF}(\text{John}, \text{John}) \& \text{WF}(\text{John}, \text{John}))$

Fig. 5d. The fourth step of the construction

STEP 5. Apply "&-rule to 8) results in 9) and 10).

1)  $(\forall x)(\text{WF}(x, 'top-manager')) \rightarrow \text{WF}(\text{John}, x)$

2)  $\neg [\exists x](\exists y)(\text{WF}(x, x) \& \text{WF}(y, x))$

3)  $\text{WF}(\text{John}, 'top-manager')$

4)  $\text{WF}(\text{John}, 'top-manager') \rightarrow \text{WF}(\text{John}, \text{John})$

5)  "WF(John, 'top-manager')" 6) WF(John, John)
X conflict with 3)

7)  $\neg [\exists y](\text{WF}(\text{John}, \text{John}) \& \text{WF}(y, \text{John}))$

8)  $\neg (\text{WF}(\text{John}, \text{John}) \& \text{WF}(\text{John}, \text{John}))$

9)  $\neg \text{WF}(\text{John}, \text{John})$
10) $\neg \text{WF}(\text{John}, \text{John})$
X conflict with 6)  X conflict with 6)

Fig. 5e. The fifth step results in a closed tableau

Since every branch of the tableau in STEP 5 is closed, we conclude that the set \{1), 2), 3\} is inconsistent.

4.4.4 Summary Remarks

1) At this stage, it is useful to clarify some of the terminology which is associated with a tableaux tree. The terminology will be used in Sect. 4.8 when we describe the construction of the legal system states. We refer to the resulting tableaux tree in STEP 5 above. We say that formulae 1), 2) and 3) constitute the root of the tree, which is at level 0 of the tree. At level 1, we have formula 4) which is the son of the root. Note that the node at level 1 consists of only one formula, i.e., formula 4). Thus we may sometimes refer to a node by the sole formula it contains. However, this convention is not always possible, e.g., the root of
the tree contains three formulae. We use the convention when no confusion can arise. By this convention we say that the two nodes at level 2 are nodes 5) and 6) respectively. Nodes 5) and 6) are sons of node 4) since they are the immediate successors of node 4), which is at level 1. Nodes 5) and 6) are said to be brothers or node 5) is a brother of node 6) and vice versa.

2) The idea behind the method of tableaux is made explicit in the following [9]:

In constructing a tableau for a set W of wffs, we think intuitively of W as a story which we are trying to criticize. At each stage of the construction, the various branches represent alternative (but more specific and detailed) versions of the same story. Moreover, if W is to be believed then at least one of the versions must be believed as well. A closed branch represents an unbelievable version.

3) Suppose that we have a branch (in some tableau) such that for the set of formulae of that branch we already possess a refutation. Then for all practical purposes that branch is as good as closed, because we know, that by successively extending it we can eventually get branches all of which are actually closed. In practice, a branch which is closed (or as good as closed) need not be extended any further even if the rules allow us to do so. For this reason, we put a cross 'X' right beneath the terminal of a closed branch. This indicates that the branch need not be extended.

4) The first order tableaux system is sound [9]. That is, if a set W of wffs is inconsistent, then there is a refutation for it [9] [33]. However, if we cannot construct a closed tableau after running the procedure for a considerably long time, we still cannot conclude that the set is consistent. An algorithm for systematically applying the above rules is given in [9]. The algorithm is claimed to be able to determine the consistency by constructing a Hintika set for a set of formulae that has not been found to be inconsistent. A Hintika set is a set of formulae which has some desirable properties: first, a Hintika set is always consistent; second, a set of formulae is always contained by its Hintika set if it exists. Clearly a set of formulae is consistent iff it can be extended to a Hintika set. Unfortunately, the algorithm developed in [9] does not stop until the inconsistency is detected. The reason for this 'ineffectiveness' is that the first order logic is semidecidable.

4.5 Restricting the Static Constraints

4.5.1 Using Many-sorted Logic

One modification to be made to the first order tableaux system is that a many-sorted logic is used. Consider the wff:
1) \((\forall x)(\exists y)\text{EARN}(x,y)\)
Every \(x\) earns some salary \(y\).

and the \(I\)- and \(V\)-rules listed in Fig.4b. Suppose we want to prove that
the above formula is inconsistent. First we apply the \(V\)-rule with \(a_0\)
replacing \(x\) throughout the subformula \((\exists y)\text{EARN}(x,y)\). The result is

1) \((\forall x)(\exists y)\text{EARN}(x,y)\)

Since the tableau is not closed, we may either apply the \(V\)-rule to 1) or
apply the \(I\)-rule to 2). Applying the \(V\)-rule to 1) will result in a
formula which is identical to 2) if we replace \(x\) by \(a_0\). This will not
lead to a closed tableau. Applying the \(I\)-rule to 2) requires that a
new constant symbol, say \(a_1\) be introduced. The result is

1) \((\forall x)(\exists y)\text{EARN}(x,y)\)

2) \((\exists y)\text{EARN}(a_0,y)\)

3) \(\text{EARN}(a_0,a_1)\)

The tableau is still not closed. If we apply the \(V\)-rule to 1) again
with \(a_0\) replacing \(x\), we will have 2) again, which does not contribute
to closing the tableau. Thus, we can either apply the \(V\)-rule to 1) with \(a_1\) replacing \(x\) or apply the \(I\)-rule to 2) with another new
constant symbol introduced. In the first case we have:

1) \((\forall x)(\exists y)\text{EARN}(x,y)\)

2) \((\exists y)\text{EARN}(a_0,y)\)

3) \(\text{EARN}(a_0,a_1)\)

4) \((\exists y)\text{EARN}(a_1,y)\)

In the second case, we have:

1) \((\forall x)(\exists y)\text{EARN}(x,y)\)

2) \((\exists y)\text{EARN}(a_0,y)\)

3) \(\text{EARN}(a_0,a_1)\)

4) \(\text{EARN}(a_0,a_2)\)

In either case, the resulting tableau is not closed. In fact, repeatedly
applying the rules will never result in a closed tableau since the formula \((\forall x)(\exists y)\text{EARN}(x,y)\) is consistent. The problem is to
find out how to stop the tableaux process once we know that further
expansion is useless. In a single-sorted logic, we have no way of
stopping the tableaux process. However, in a many-sorted logic, we can
take the advantage that the variable \(x\) ranges over persons and the
variable \(y\) ranges over money and, the set of persons and the set of
values of money are disjoint.

Simply speaking, a many-sorted logic is like the ordinary first order
logic except that each constant or variable belongs to some sort.
Many-sorted logic has been used by some authors in information
modeling [84] [22] as well as in database query answering [53] [76]. Space limitation does not allow us to present a formal definition of many-sorted logic, the reader is referred to [22] [84].

Thus applying the \( \forall \)-rule to 1) with \( a_1 \) replacing \( x \) is meaningless and unnecessary since \( a_1 \) was introduced as a constant symbol of sort money. Thus, after we have

1) \( \forall x (\exists y \text{EARNS}(x, y)) \)
2) \( \exists y \text{EARNS}(a_0, y) \)
3) \( \text{EARNS}(a_0, a_1) \)

we can stop the process since \( \text{EARNS}(a_0, a_1) \) is a model for the formula \( (\forall x (\exists y \text{EARNS}(x, y)) \) which in a many-sorted logic, says that for every person \( x \) there exists a \( y \) which is of sort money such that \( x \) earns \( y \). Since we have only one person \( a_0 \) in the universe of discourse, \( \text{EARNS}(a_0, a_1) \) satisfies formula 1).

4.5.2 Excluding Recursiveness

Suppose that the static constraints are specified in a many-sorted logic, we still have problems. Consider the following wff:

\[ \text{sc: } (\forall x)(\exists y \text{WF}(x, y)) \]

where \( \text{WF}(x, y) \) represents that person \( x \) works for person \( y \). That is, \( \text{WF} \) is a binary predicate of sort (person, person).

In order to conclude that \( (\forall x)(\exists y \text{WF}(x, y)) \) is consistent, we have to construct a model \( S[U, I] \) for it. For simplicity, we suppose that the universe consists of only one sort person. Since the universe \( U \) of the model is, by definition, non-empty, we must have at least one person, say \( a_0 \in U \). Further, since the formula says that for every person \( x \) in \( U \) there exists a person \( y \) in \( U \) such that \( x \) works for \( y \) (or \( \text{WF}(x, y) \)) is true. Two possibilities are there:

1) The \( \text{WF} \) relation does not allow the following situation:
   
   For any sequence of element \( a_{i1}, \ldots, a_{in} \) in \( U \)
   
   \( a_{i1} \) works for \( a_{i2} \)
   \( a_{i2} \) works for \( a_{i3} \)
   
   : 
   
   \( a_{in-1} \) works for \( a_{in} \)
   \( a_{in} \) works for \( a_{i1} \)

   In this case, it is obvious that \( U \) must be infinite.

2) The \( \text{WF} \) relation allows the above situation, in this case, \( U \) will be finite. For example, a model \( S[U, I] \) for the formula \( \text{sc} \) above will have

   \[ U = \{ a_0 \} \text{ and } I = \{ \text{WF}(a_0, a_0) \} \]

   In the first case, it is impractical to construct any model for the
formula sc and hence we do not allow such formula to appear in the static constraints. In the second case, we use the expression $(\exists y)WF(a_i,y)$ instead of $(\exists y)WF(a_i,y)$ to indicate that $y$ ranges over a finite universe of discourse.

The above discussion indicates that the first order tableaux system cannot be used directly. We have to modify the rules so that sorted symbols and the finiteness of the universe of discourse are taken into account. This will be described in the sequel.

4.6 Detection of Recursiveness

4.6.1 Prenex Form

The recursive relation discussed in the last subsection can be represented by a directed graph called a $\beta$-graph.

In order to construct such a graph, we first translate the static constraints into prenex form.

The translation of a formula into its prenex form can be done by repeatedly applying the following equivalences [98] [9]:

1) $(\exists x)P(x) \lor (\exists x)Q(x) \iff (\exists x)\{ P(x) \lor Q(x) \}$
2) $(\forall x)P(x) \land (\forall x)Q(x) \iff (\forall x)\{ P(x) \land Q(x) \}$
3) $(\forall x)P(x) \lor Q \iff (\forall x)\{ P(x) \lor Q \}$
4) $(\exists x)P(x) \land Q \iff (\exists x)\{ P(x) \land Q \}$
5) $(\forall x)P(x) \rightarrow Q \iff (\exists x)\{ P(x) \rightarrow Q \}$
6) $(\exists x)P(x) \rightarrow Q \iff (\forall x)\{ P(x) \rightarrow Q \}$
7) $P \rightarrow (\forall x)Q(x) \iff (\forall x)\{ P \rightarrow Q(x) \}$
8) $P \rightarrow (\exists x)Q(x) \iff (\exists x)\{ P \rightarrow Q(x) \}$
9) $\neg (\exists x)P(x) \iff (\forall x)\neg P(x)$
10) $\neg (\forall x)P(x) \iff (\exists x)\neg P(x)$

Note that alphabetic changes of some of the bounded variables in the formula that is to be translated are needed [9]. Particularly, alphabetic changes are required by the applications of equivalences 3) to 8) above. It can be proved that such alphabetic changes will not alter the meaning of the formula [9].

Since the above translation can always be done, we will assume in the following that the static constraints are in prenex form.
4.6.2 The D-graph

A directed graph is an ordered pair \(<V, E>\), where \(V\) is a finite set of nodes and \(E \subseteq V \times V\) is a set of arcs. Arc \(<v_i, v_j>\) goes from \(v_i\) to \(v_j\). A path is a sequence \(v_0, \ldots, v_n\) of nodes \((n \geq 0)\) such that \(<v_i, v_{i+1}>\) is an arc for \(i = 0, \ldots, n-1\), and such that the \(v_i\)'s are distinct; except that we allow \(n > 0\) and \(v_0 = v_n\), in which case the path is called a cycle.

The directed graph (D-graph) for detecting the recursive relation discussed in the last subsection can be constructed as follows. The nodes of the graph are labeled by the sorts and an arc goes from node \(i\) to node \(j\) iff there is an occurrence of \((\forall x)\ldots(\exists y)\) or \((\exists! y)\ldots(\exists y)\), where \(x, y\) are variables of sorts \(i\) and \(j\) respectively.

There is a recursive relation (in the sense discussed in the last subsection) iff there is a cycle in the D-graph. If there is a cycle in the D-graph for the set of static constraints, then the modified tableaux approach is not applicable. Some decisions must be made to replace some of the \(\exists\)-quantifiers by \(\exists!\)-quantifiers and repeat the process again. Such a restriction is too strong. In Chapter 5, we will discuss another restriction which is the weakest in the sense that further reduction of the weakest restriction will result in a class of formulae its consistency is undecidable.

The D-graph for the many-sorted formula \((\forall x)\exists y\text{EARN}(x,y)\) with \text{EARN} being of sort \(\text{person, money}\) has the directed graph as depicted below.

\[
\text{person} \quad \overset{0}{\longrightarrow} \quad \text{money}
\]

Since there is an occurrence of \((\forall x)(\exists y)\) with \(x\) of sort \text{person} and \(y\) of sort \text{money}.

The D-graph for the many-sorted formula \((\forall x)(\exists y)\text{WF}(x,y)\), on the other hand, has a loop that goes from sort \text{person} to sort \text{person}:

\[
\text{person} \quad \overset{0}{\longrightarrow} \quad \overset{1}{\longrightarrow} \quad \text{person}
\]

Since there is a cycle in the D-graph, the formula \((\forall x)(\exists y)\text{WF}(x,y)\) is not allowed to be a static constraint.

4.7 The Modified Tableaux Approach

4.7.1 The Modified Deduction Rules

The presumption of the modified tableaux approach is that the D-graph for the set of static constraints contains no cycle. We also assume that the static constraints are in prenex form due to the transformation presented in Sect. 4.6.1.

In what follows, we denote by \(s(c)\) the sort of constant symbol \(c\), and by \(s(x)\) the sort of variable \(x\). \(\text{PS}(k)\) denotes the parameter space of sort \(k\), which is the set of individuals or 'representatives' of sort \(k\) introduced so far by the modified tableaux approach. Initially, \(\text{PS}(k)\)
is empty for all \( k \). Elements of a parameter space will be denoted as \( a_0, \ldots, a_n, b_0, \ldots, b_m \), etc. The notion of parameter space is borrowed from [33].

In this section we assume that no function symbol appears in the formulae. It is shown in [9] that function symbols can be eliminated in a logic with identity.

The modified tableaux approach uses the following quantificational rules, where \( w(x) \) denotes a formula in which \( x \) is free:

<table>
<thead>
<tr>
<th>Rule</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>((\exists x)w(x))</td>
<td>From ((\exists x)w(x)) infer ( w(a_i) ) by introducing a new element ( a_i ) into ( PS(s(x)) ).</td>
</tr>
<tr>
<td>((\forall x)w(x))</td>
<td>If ( PS(s(x)) = {} ) then introduce a new element of sort ( s(x) ) into ( PS(s(x)) ). From ((\forall x)w(x)) and ( PS(s(x)) = { a_0, \ldots, a_n } ) infer ( w(a_0), \ldots, w(a_n) ).</td>
</tr>
<tr>
<td>((\exists! x)w(x))</td>
<td>If ( PS(s(x)) = {} ) then introduce a new element of sort ( s(x) ) into ( PS(s(x)) ). From ((\exists! x)w(x)) and ( PS(s(x)) = { a_0, \ldots, a_n } ) infer ( w(a_0) ) or ( w(a_n) ).</td>
</tr>
</tbody>
</table>

**Fig. 8. The redefined quantificational rules**

From now on, the \( V \)-rule, \( I \)-rule and \( III \)-rule refer to the three rules that are defined in Fig.6 but not those presented in Fig.4b.

The propositional deduction rules that are used by the modified tableaux approach include the \&-rule, \( V \)-rule, \( ^\rightarrow \)-rule, \( \& \)-rule, \( V \)-rule, \( ^\rightarrow \)-rule as depicted in Fig.4a. We also use the SI-rule (self-identity), SF-rule (substitute in functions), SP-rule (substitute in predicates) as defined in Sect.4.4.1.

### 4.7.2 The Modified Method

We will use the following set of constraints to illustrate the method.

- \( sc_1 : (\forall x)(\exists y) [ M(x) \rightarrow E(x,y) ] \)
  - Every manager is an employee.
- \( sc_2 : (\forall x)(\forall y) [ E(x,y) \rightarrow y > 20000 ] \)
  - Every employee earns more than 20000.
where \( E \) is of sort \( \{ \text{person, salary} \} \), \( M \) is of sort \( \{ \text{person} \} \) and \( > \) is the arithmetic comparison relation. We assume that the following axioms are defined for \( > \):

- irreflexive: \(~(x>x)\)
- asymmetric: \( x>y \rightarrow ~(y>x) \)
- transitive: \( x>y \land y>z \rightarrow x>z \)

The modified tableaux approach is described as follows.

1) Denote the set of static constraints by \( T_0 \), which is the root of the tableau. \( T_0 \) can also be seen as the tableau that is constructed so far. For each sort \( k \) defined in the language, define a parameter space \( PS(k) \) of sort \( k \). Initially all \( PS(k) \)'s are empty.

After applying this step to the example we have:

\[
\begin{align*}
(\forall x) (\exists y) ( M(x) \rightarrow E(x,y) ) \\
(\forall x) (\forall y) ( E(x,y) \rightarrow y > 20000 )
\end{align*}
\]

and

\[
PS(\text{person}) = \{ \}, \quad PS(\text{salary}) = \{ \}
\]

2) Collect every constant symbol occurring in the static constraints in its respective parameter space. After this step we have:

\[
\begin{align*}
(\forall x) (\exists y) ( M(x) \rightarrow E(x,y) ) \\
(\forall x) (\forall y) ( E(x,y) \rightarrow y > 20000 )
\end{align*}
\]

and

\[
PS(\text{person}) = \{ \}, \quad PS(\text{salary}) = \{ 20000 \}
\]

3) Apply the \( I \)-rule to the tableau already constructed by introducing a new element into the respective parameter space. The formula to which the \( I \)-rule was applied is marked as "used". In this section, we will use \( 'Z' \) to mark a used formula. This process is repeated until no more application of the \( I \)-rule is possible.

The result for our example is the same as that from the last step since the \( I \)-rule cannot be applied to the tableau already obtained.

4) In the following steps, if a branch is found to contain a ground instance of a predicate and the negation of that ground instance, the branch is immediately closed. We will put a cross \( 'X' \) right beneath the tip of the branch to indicate that the branch is closed.

5) Expand any formula which contain a \( \forall, \exists \) or \( ! \) quantifier in the tableau first. The following method is used to extend the tableau.

- When a formula \((\forall x)w(x)\), where \( x \) is of sort \( k \), is expanded and \( PS(k) = \{ a_0, a_1, \ldots, a_n \} \), we attach \( w(a_0), w(a_1), \ldots, w(a_n) \) consecutively to each branch of the tableau to which \((\forall x)w(x)\) belongs. If \( PS(k) = \{ \} \), we introduce a new element into \( PS(k) \) and expand \((\forall x)w(x)\) accordingly. After this our example becomes

\[
\begin{align*}
(\forall x) (\exists y) ( M(x) \rightarrow E(x,y) ) & \quad \text{Z} \\
(\forall x) (\forall y) ( E(x,y) \rightarrow y > 20000 ) & \\
(\exists y) ( M(a_0) \rightarrow E(a_0,y) ) \\
\text{and} \quad PS(\text{person}) = \{ a_0 \}, \quad PS(\text{salary}) = \{ 20000 \}
\end{align*}
\]
- When a formula (∃y)w(y) where y is of sort k, is expanded. We introduce a new element aᵢ into PS(k) and attach w(aᵢ) to each branch of the tableau to which (∃y)w(y) belongs. That is

\[
(\forall x) \ (\exists y) \ (M(x) \rightarrow E(x,y)) \quad \exists \chi
\]
\[
(\forall x) \ (\forall y) \ (E(x,y) \rightarrow y > 20000)
\]
\[
(\exists y) \ (M(a₀) \rightarrow E(a₀,y)) \quad \exists \chi
\]
\[
M(a₀) \rightarrow E(a₀,b₀)
\]
and \(PS(person) = \{ a₀ \}, \ PS(salary) = \{ 20000, b₀ \}\)

- When (∀y)w(y) is expanded where y is of sort k and PS(k) = \{ b₀, b₁, ..., bₘ \}, we attach

\[
\begin{array}{c}
w(b₀) \quad \vdots \\
w(b₁) \\
\vdots \\
w(bₘ)
\end{array}
\]

to each branch of the tableau to which (∀y)w(y) belongs.

(Our example cannot show this.)

The above process is repeated until no unused formula in the tableau contains quantifiers.

After expanding all the quantified formulae we have only quantifier-free formulae in the tableau which have not been used. Our example after this step is

\[
(\forall x) \ (\exists y) \ (M(x) \rightarrow E(x,y)) \quad \exists \chi
\]
\[
(\forall x) \ (\forall y) \ (E(x,y) \rightarrow y > 20000) \quad \exists \chi
\]
\[
(\exists y) \ (M(a₀) \rightarrow E(a₀,y)) \quad \exists \chi
\]
\[
M(a₀) \rightarrow E(a₀,b₀)
\]
\[
(\forall y) \ (E(a₀,y) \rightarrow y > 20000) \quad \exists \chi
\]
\[
E(a₀,20000) \rightarrow 20000 > 20000
\]
\[
E(a₀,b₀) \rightarrow b₀ > 20000
\]

Note that the parameter spaces do not grow after all of the existential quantifiers have been expanded.

6) Expand the quantifier-free formulae in a depth-first fashion. That is, the leftmost open branch is always expanded first. From now on a formula that has been used will be marked differently, say by \(\exists'\). Several cases are considered in this step

- If the leftmost branch is closed before all the unmarked formulae have been used, we remove all the \(\exists'\) marks and expand the next leftmost open branch.

- If all the branches are closed, the set of constraints is inconsistent.
If no expansion of the leftmost branch is possible and the leftmost branch is not closed, then the set of constraints is consistent.

\[(\forall x) (\exists y) \ (M(x) \rightarrow E(x,y)) \quad I\]
\[(\forall x) (\forall y) \ (E(x,y) \rightarrow y > 20000) \quad I\]
\[(\exists y) \ (M(a_0) \rightarrow E(a_0,y)) \quad I\]
\[M(a_0) \rightarrow E(a_0,b_0) \quad I'\]
\[(\forall y) \ (E(a_0,y) \rightarrow y > 20000) \quad I\]
\[E(a_0,20000) \rightarrow 20000 > 20000 \quad I'\]
\[E(a_0,b_0) \rightarrow b_0 > 20000 \quad I'\]

\[\mu_2 = \{\neg M(a_0), E(a_0,b_0)\}\]
\[\mu_1 = \{\neg E(a_0,20000)\}\]
\[X \ (\text{conflicts with the irreflexive axiom})\]
\[\mu_0 = \{E(a_0,b_0), b_0 > 20000\}\]

**Fig. 7. The final tableaux tree for \{ sc1, sc2 \}**

Fig. 7 shows the result after expanding the quantifier-free formulae. Since no further expansion is possible and the tableau is still not closed, we may conclude that the set \{ sc_1, sc_2 \} is consistent. Note that in Fig. 7, we have shown three sets \mu_i, i = 0, 1, 2. These sets are called associate sets which will be defined in Sect. 4.8.1.

### 4.7.3 Summary Discussions

1) In practice, the modified tableaux approach can be made more efficient by associating more information with the D-graph. For example, an arc going from node i to node j may be labeled by a list of formula numbers. A formula number is in the list labeling the arc from node i to node j if the prenex of the formula contains \((\forall x)\)...(\exists y) where x, y are of sort i and j respectively. If there are n such occurrences in the formula, we allow the formula number to appear n times in the list (alternatively, we may associate a counter with each formula number). When the tableau is being constructed, we always look for an unused formula, in the latest constructed tableau, whose leftmost quantifier is an existential quantifier \(\{x\}\), ..., expand the formula first. When there is no such formula in the tableau, then since the D-graph contains no cycle, there must be some node (or nodes) with no arc coming in. In graph theory terminology,
there must be some node of indegree zero. Let the set of such nodes be denoted as \{k_1, k_2, ..., k_m\}. Moreover, at least one of the unused formulae must have its leftmost quantifier being a \forall-quantifier or \exists!-quantifier which quantifies a variable of sort \(k_i\) for some \(i\) such that \(1 \leq i \leq m\). Expand this formula and remove one occurrence of the formula number from each of the list associated with each of the arcs going out from node \(k_i\). When all the formula numbers labelling an arc have been removed, we remove the arc immediately. In this way, we may introduce as few new elements as possible into the parameter spaces.

2) It may happen that the D-graph consists of several subgraphs such that there exists no arc from a node of one subgraph to a node of another subgraph. In this case, the set of subgraphs partitions a subset of the static constraints into equivalence classes. Each equivalence class corresponds to a subgraph. That is, a static constraint is in an equivalence class iff the constraint is in a list labelling an arc of the subgraph corresponding to the equivalence class. We say that the subgraphs partition a subset of the static constraints because some of the static constraints may not appear in the lists labelling the arcs of the subgraphs. For example, \(sc_1, sc_2\) above have the following D-graph:

```
person o-----------------o salary
        \--------\     \--------\     \--------\     \--------\
sc_1     \--------\ sc_2     \--------\ sc_3     \--------\ sc_4
```

Note that \(sc_2\) does not appear in any list labelling any arc.

If we augment each equivalence class with those static constraints which are not in any of the equivalence classes, we have a set of coverings. It can be easily proved that the set of static constraints is consistent iff each covering is consistent. That is, the model for the set of the static constraints is the union of the submodels of the coverings. Thus, we may consider each of the coverings separately using the method described in point 1).

3) To facilitate the model construction which is to be described in the next section, the modified tableaux approach may also use the following propositional deduction rules:

\[
\begin{align*}
\&-rule: & (w_1 \& w_2) \rightarrow w_3 \\
\rightarrow \&-rule: & w_1 \rightarrow (w_2 \& w_3)
\end{align*}
\]
Some other rules can be similarly defined. We will not consider these further here. Note that some suitable ordering for applying the propositional rules should be defined after the above rules have been introduced. For example, when a formula of the form \((w_1 \& w_2) \rightarrow w_3\) is to be expanded, we will use the \(\rightarrow\)-rule instead of the \(\&\)-rule.

4.8 Construction of System States

4.8.1 The Associate Sets

For each level from the bottom of the tableaux tree (see Fig.7), we define an associate set \(\mu_i\) as follows. The associate set for the bottom level is denoted as \(\mu_0\). The associate set for the next level is denoted as \(\mu_1\), etc. Note that the \(\mu_i\)'s are numbered in an opposite way in which the levels of a tree are numbered. That is, the root of a tree is usually defined to be at level \(0\), the bottom level of the tree is of level \(n\) for some \(n>0\) which is usually defined as the height of the tree. Given a tree of height \(n\), the level at which an associate set \(\mu_i\) is constructed is \(j = n-i\). In what follows, an atomic formula or its negation is called a prime. The ground instance of a prime is called a prime instance. E.g., \(M(a_0), 20000 < 20000\) are prime instances.

The associate set \(\mu_i\), contains the prime instance appearing in the leftmost open path as well as its brothers that is not in conflict with a system axiom (if any). This idea is illustrated in the following figure, where we assume that \(P_1, P_2, \ldots\) represent prime instances which do not conflict with any system axioms, while \(F\) denotes any prime instance that is in conflict with some system axioms. Recall that an 'X' beneath a node indicates that the branch is closed.
The tableaux tree that is obtained in the last section at level \( j = n - i \)

level 0 The root

\[ \cdots \]

level \( j \) 

\[ P_1 \quad F \quad P_2 \quad \ldots \quad P_k \]

\[ X \quad X \quad \cdots \quad X \]

level \( n \)

\[ P_{k+1} \quad F \quad P_{k+2} \]

\[ X \quad X \quad \cdots \quad X \]

\[ \mu_i = \{ P_1, P_2, \ldots, P_k \} \]

\[ \mu_0 = \{ P_{k+1}, P_{k+2} \} \]

Fig. 8. The associate sets for a tableaux tree

As a concrete example, the tree in Fig.7 has the following associate sets:

- \( \mu_0 = \{ \neg E(a, b_0), b_0 > 20000 \} \)
- \( \mu_1 = \{ \neg E(a, 20000) \} \), note that 20000 > 20000 conflicts with the system axiom \( \neg(x > x) \).
- \( \mu_2 = \{ \neg M(a), E(a, b_0) \} \)

4.8.2 The Semantics of the Associate Sets

Intuitively, the associate set \( \mu_i \) contains the prime instances \( P_1, \ldots, P_k \) associate with some quantifier-free formula \( w \) at some level \( j < n - i \) such that \( w \) is true iff \( P_1 \lor P_2 \lor \ldots \lor P_k \) is true. This follows immediately from the semantics of the deduction rules. Furthermore, since \( w \) is deduced from some static constraint \( sc \), the semantics of the deduction rules implies that \( sc \) holds only if \( w \) holds. In general, there may be more than one \( w \) that is deduced from \( sc \) and all of them are in the same path leading from the root to some leave of the tree. For example, the wffs

\[ \mu_1: \quad E(a, 20000) \rightarrow 20000 > 20000 \]

\[ \mu_2: \quad E(a, b_0) \rightarrow b_0 > 20000 \]

in Fig.7 are deduced from \( sc_2 \). They appear in the same path from the root to some leave of the tree. Thus, if all such \( w \)'s hold then \( sc \) holds. Moreover, any system state \( S \) which satisfies \( sc \) must satisfy \( P_1 \lor P_2 \lor \ldots \lor P_k \), which is the disjunction of the elements of the associate set \( \mu_i \) at level \( n - i \). In this sense, we see that \( \mu_i \) is associated with some static constraint \( sc \). However, \( S \) satisfies \( P_1 \lor P_2 \lor \ldots \lor P_k \) iff \( S \) satisfies at least one \( P_m, 1 \leq m \leq k \).

In set-theoretical terms, let \( \mu_i = \{ P_1, \ldots, P_k \} \) be the associate set of \( sc_q \) for some \( q \), where \( sc_q \) is a static constraint. Then a system state \( S \) in which \( sc_q \) may hold will contain at least one \( P_m \) as described above. Since there are \( 2^k \) subsets of \( \mu_i \) and the empty subset does not
contain any \( P_m \) for all \( m \), we have \( 2^k - 1 \) system states in which \( \mu^0 \) holds. For example, \( \mu^0 = \{ \neg E(a_0, b_0), b_0 > 20000 \} \) is the associate set for the static constraint

\[
\text{sc}_2: \quad (\forall x)(\forall y)(\ E(x, y) \implies y > 20000 )
\]

which is true only if \( E(a_0, b_0) \implies b_0 > 20000 \) is true, or only if \( \neg E(a_0, b_0) \lor b_0 > 20000 \) is true. It is easily seen that

\[
\neg E(a_0, b_0) \lor b_0 > 20000
\]

is true if

1) both \( \neg E(a_0, b_0) \) and \( b_0 > 20000 \) are true; or

2) \( \neg E(a_0, b_0) \) is true and, \( b_0 > 20000 \) is not true (i.e., \( \neg (b_0 > 20000) \) is true); or

3) \( \neg E(a_0, b_0) \) is not true (i.e., \( E(a_0, b_0) \) is true) and \( b_0 > 20000 \) is true.

Thus, we know that there are three combinations of the elements of \( \mu^0 \) in which \( \text{sc}_2 \) may hold:

- combinatin 1: \( \{ \neg E(a_0, b_0), b_0 > 20000 \} \)
- combinatin 2: \( \{ E(a_0, b_0), b_0 > 20000 \} \)
- combinatin 3: \( \{ \neg E(a_0, b_0), \neg (b_0 > 20000) \} \)

In other words, we may say that there are three alternatives of assigning the truth values to the primes of \( \mu^0 \) such that \( \text{sc}_2 \) may be true. We cannot conclude that \( \text{sc}_2 \) must hold in these states since, from Fig.7, we know that \( \mu^1 \) is also associated with \( \text{sc}_2 \). Thus, in order to have \( \text{sc}_2 \) held in a system state \( S \), it is required that \( S \) must satisfy one of the combinations above and the sole combination of \( \mu^1 \) which is \( \neg E(a_0, 20000) \).

In model theoretical terms,

\[
\{ \neg E(a_0, 20000) \} \cup \{ \text{one of the alternatives} \}
\]

must be consistent. If there is such a (consistent) union, then this union is a model for \( \text{sc}_2 \). Otherwise, \( \text{sc}_2 \) is not consistent.

### 4.8.3 Formal Definition of Alternative Sets

The above discussion is formalized as follows.

Let power-set(\( \mu_i \)) be the power set of \( \mu_i = \{ P_1, \ldots, P_k \} \), for \( i = 0, 1, \ldots \), and, \( Z_{ij} \), \( j = 0, \ldots, 2^k - 2 \), be an element of power-set(\( \mu_i \)) - [ ]. To each such \( Z_{ij} \) we define an alternative set \( A_{ij} \) as follows:

- \( P_m \) is in \( A_{ij} \) iff \( P_m \) is in \( Z_{ij} \) and \( \mu_i \), where \( m = 1, \ldots, k \).
- Pₐ is in A₁j iff Pₐ is in μᵢ but not in ⁵₁j, where m = 1, ..., k.
- No other elements are in A₁j besides those defined in the last two paragraphs.

4.8.4 An Example

The alternative sets for our example are shown below:

For μ₀, we have

A₀₀ = \{ E(a₀, b₀), b₀ > 20000 \}
A₀₁ = \{ E(a₀, b₀), b₀ > 20000 \}
A₀₂ = \{ E(a₀, b₀), b₀ > 20000 \}

For μ₁, we have

A₁₀ = \{ E(a₀, 20000) \}

For μ₂, we have

A₂₀ = \{ M(a₀), E(a₀, b₀) \}
A₂₁ = \{ M(a₀), E(a₀, b₀) \}
A₂₂ = \{ M(a₀), E(a₀, b₀) \}

4.8.5 Construction of Legal System States

Let Aᵢ denote the set of all the alternative sets of μᵢ. We have for example A₀ = \{ A₀₀, A₀₁, A₀₂ \}, A₁ = \{ A₁₀ \} and A₂ = \{ A₂₀, A₂₁, A₂₂ \}. That is, Aᵢ denotes the alternatives of assigning the truth values to the primes such that the static constraint with which μᵢ is associated may be true. From Fig.7 we see that μ₀ and μ₁ are associated with the static constraint sc₂. As discussed earlier, in order to show that sc₂ is consistent, we must show that there is an Aᵢᵢ, i=0,1,2, such that A₀₀ U A₁₀ is consistent. Fortunately, each of the A₀₀ᵢ's is consistent with A₁₀ and hence there are three states for sc₂. However, we want sc₁ and sc₂ to be held at the same time, this means that we must union the three states with each of the three alternatives of μ₂ and regard a consistent union as a system state in which sc₁ and sc₂ will hold. An inconsistent union implies that it cannot make sc₁ and sc₂ true at the same time. This is formalized as below.

Let the Cartesian product of all the Aᵢᵢ's be denoted as:

n
X Aᵢ = A₀ X A₁ X ... X Aᵦ
i=0

where n is the number of μᵢᵢ's.

For each C in X Aᵢ, we form the union of all the components of C. If the union is consistent then it is a legal system state (i.e. a model) for the set of static constraints; otherwise we discard it. By this
method we can construct four system states for our example:

- \( S_0 = \{ \neg E(a_0, b_0), \neg b_0 > 20000, \neg M(a_0), \neg E(a_0, 20000) \} \)
- \( S_1 = \{ \neg E(a_0, b_0), \neg b_0 > 20000, \neg M(a_0), \neg E(a_0, 20000) \} \)
- \( S_2 = \{ E(a_0, b_0), b_0 > 20000, \neg M(a_0), \neg E(a_0, 20000) \} \)
- \( S_3 = \{ E(a_0, b_0), b_0 > 20000, M(a_0), \neg E(a_0, 20000) \} \)

The reader can easily verify that these are the only system states in which \( sc_1 \) and \( sc_2 \) are true.

It can be proved that any system state that is constructed from the above method will satisfy all the static constraints. Further, any set of static constraints which satisfy the restriction imposed in Sect. 4.5 and Sect. 4.6 and which is consistent will have at least one system state. The formulation of the proof requires that the modified tableau approach be reformulated, since the present formulation is suitable only for explanation but not for proof. For this reason, we will not prove the above statements in this thesis.

4.9 Analysis of Operations

4.9.1 A Short Note on Unification

We first briefly review the concept of unification from artificial intelligence [178] [172].

Simply speaking, unification is a process which results in a substitution of terms to variables from a set of primes such that the use of the substitution collapses the set to a singleton. For example, the substitution which collapses the set of primes \( \{ E(x, z), E(x, b_0) \} \) is \( \Theta = \{ <b_0, z> \} \) which means substituting \( b_0 \) for variable \( z \) throughout the set of primes. The use of this substitution \( \Theta = \{ <b_0, z> \} \) to the set of primes above collapses the set to a singleton \( \{ E(x, b_0) \} \). This notion is denoted as \( \{ E(x, z), E(x, b_0) \} \Theta = \{ E(x, b_0) \} \).

In general, to collapse a set of primes to a singleton we need to substitute a number of terms for a number of variables. Thus, a substitution in general can be represented by a set of ordered pairs \( \Theta = \{ <t_1, v_1>, ..., <t_n, v_n> \} \).

It is required that each occurrence of a variable \( v_i \) has the same term \( t_i \) substituted for it; that is, \( v_i = v_j \) implies \( i = j \) for \( i, j = 1, ..., n \). Moreover, if \( <t_i, v_i> \) is in \( \Theta \), then \( t_i \) cannot contain \( v_i \) for all \( i = 1, ..., n \). A formal presentation of the unification process is included in Appendix B. Here we consider that the above informal presentation is sufficient for our purposes.

4.9.2 Informal Introduction to Operation Analysis

We now present the method of analysing of the operations. We analyse each of the operations to see if it is applicable in some legal information system state. Intuitively, an operation \( \langle op \rangle \) is applicable
in a legal system state $S$ iff $S \vdash \text{pre}(\text{op}) \theta$ for some substitution

$\theta = \{ \langle t_1, \pi_1 \rangle, \ldots, \langle t_n, \pi_n \rangle \}$

where for $i = 1, \ldots, n$, $t_i$ is a term of $S$ and $\pi_i$ is a parameter of the precondition $\text{pre}(\text{op})$ of the operation $\langle \text{op} \rangle$. The idea behind $S \vdash \text{pre}(\text{op}) \theta$ can be explained as follows. We regard $\langle \text{op} \rangle$ as a procedure which is described declaratively, $\text{pre}(\text{op}) \theta$ is regarded as a procedure call to the procedure $\langle \text{op} \rangle$ with actual parameters $t_i$ substituted for formal parameters $\pi_i$ for $i = 1, \ldots, n$. Clearly, the procedure call can be performed in state $S$ iff $S \vdash \text{pre}(\text{op}) \theta$.

After performing the operation, the system enters a new state $S' = S(\langle \text{op} \rangle, S)$ which must satisfy the postcondition of the operation, i.e., $S' \vdash \text{post}(\langle \text{op} \rangle) \theta$. The idea behind $S' \vdash \text{post}(\langle \text{op} \rangle) \theta$ can be explained as follows. We regard $\text{post}(\langle \text{op} \rangle)$ as specifying some condition which must hold when any execution of the procedure $\langle \text{op} \rangle$ is successfully terminated. Thus, $\text{post}(\langle \text{op} \rangle) \theta$ represents the condition which is expected to hold when a call to $\langle \text{op} \rangle$ with $t_i$'s as the actual parameters is successfully executed.

Finally, the transition from $S$ to $S'$ should preserve those information that was true in the old state but is not changed by the operation. This is called the preservability analysis of the operation. It can be regarded as a test to prevent unwanted side effects.

From a logic point of view, a system state $S$ that is constructed in Sect. 4.8.5. consists of a set of ground primes which are WFFs. To show that $S \vdash w$ it suffices to show that $S \cup \{ "w" \}$ is inconsistent. That is, $S \cup \{ "w" \}$ has a closed tableau. Thus, to prove that $S \vdash \text{pre}(\langle \text{op} \rangle) \theta$ is equivalent to proving that $S \cup \{ \text{pre}(\langle \text{op} \rangle) \theta \}$ is inconsistent. In a similar way, we can prove $S' \vdash \text{post}(\langle \text{op} \rangle) \theta$.

Note that the temporal assertion in an operation description concerns the history of the "databas". That is, when we consider whether each of the operations can ever be executed, we may ignore the temporal assertions of the operations. We take into account the temporal assertions when we consider sequences of operations (see Sect. 4.10).

### 4.9.3 Formal Analysis of Insert and Delete Operations

The above discussion leads to the following framework for analysing the 'insert' and 'delete' operations. We assume in this thesis that an insert (delete) operation changes one or more prime formulae from false to true (true to false). An update operation changes the values of some of the terms of prime formulae.

For an 'insert' or 'delete' operation $\langle \text{op} \rangle$ we check for applicability, acceptability and preservability:

1) There is some system state $S_i$ (i.e., the prestate) such that there is a closed tableau for $\{ \text{pre}(\langle \text{op} \rangle) \theta \} \cup S_i$ for some $\theta$ as defined in Sect. 4.9.2. That is, we check the applicability of the operation.

2) There is some state $S_j$ (i.e., the poststate) such that there is a closed tableau for $\{ \text{post}(\langle \text{op} \rangle) \theta \} \cup S_j$ where $\theta$ is as in 1). That is, we check the acceptability of the operation.
3) Those ground primes that are not changed by the operation must be preserved. Let \( P = \text{pre}(\langle op \rangle) \) and \( Q = \text{post}(\langle op \rangle) \), and \( \alpha \) denote any ground instance of a prime formula. We check

\[
S_1 - ([\alpha:P|\alpha]) \cup ([\alpha:Q|\alpha] - \{\alpha:P|\alpha\}) = S_j
\]

where \( S_i \) and \( S_j \) denote the prestate and the poststate of the operation \( \langle op \rangle \) respectively. \( Q|\alpha \) means that \( \alpha \) is a logical consequence of \( Q \) (see Sect.3.1.2).

The meaning of the above formula can be explained as follows:

\([\alpha:P|\alpha] \cup \{\neg \alpha:Q|\alpha\}\)

denotes the "deleted statements". It represents those ground primes that are specified to be true by the precondition and falsified by the postcondition.

\([\alpha:Q|\alpha] - \{\alpha:P|\alpha\}\)

denotes the "added statements". It represents those ground primes that are specified to be true by the postcondition but not specified to be true by the precondition. Thus, the formula in 3) says that the prestate minus the "deleted statements" and then plus the "added statements" must equal to the poststate.

Note that we cannot use \([\alpha:P|\alpha] - \{\alpha:Q|\alpha\}\) to represent the "deleted statements" because this expression denotes those ground primes that are specified to be true by the precondition but not specified to be true by the postcondition. However, those that are not consequences of the postcondition can be divided into two groups: the first group consists of those each of which is not consistent with the postcondition, i.e., the negation of each of them is a consequence of the postcondition; the second group consists of those each of which is consistent with the postcondition but it is not a consequence of the postcondition. Clearly, \([\alpha:P|\alpha] \cup \{\neg \alpha:Q|\alpha\}\) is a subset of \([\alpha:P|\alpha] - \{\alpha:Q|\alpha\}\). The former denotes the "deleted statements" but the latter contains more than the "deleted statements".

4.9.4 The Description of an Example

For illustration of the analysis of the operations, we assume the two static constraints \( sc_1 \), \( sc_2 \) as defined in Sect.4.7.2:

\( sc_1 : (\forall x)(\exists y)(M(x) \rightarrow E(x,y)) \)

Every manager is an employee.

\( sc_2 : (\forall x)(\forall y)(E(x,y) \rightarrow y > 20000) \)

Every employee earns more than 20000.

Moreover, we assume the following operation descriptions:

\[ \text{hire}(\overline{x}): \text{always} \neg (\exists y)E(\overline{x},y) \]

\[ \Rightarrow \delta(\text{hire}(\overline{x}),S_i) \vdash (\exists y)(E(\overline{x},y) \& y > 20000) \]

\[ \text{fire}(\overline{x}): \text{ax} \& S_i \vdash (\exists y)(E(\overline{x},y) \& y > 20000) \& \neg M(\overline{x}) \]

\[ \Rightarrow \delta(\text{fire}(\overline{x}),S_i) \vdash \neg (\exists y)E(\overline{x},y) \]

\[ \text{raise}^* (\overline{x}, 10\% y): \text{ax} \& S_i \vdash E(\overline{x},y) \]


\[ \delta(\text{raise}^\delta(x, 100\cdot y), S_1) \models E(x, y + 100\cdot y) \]

**promote** (x): \( \forall y, S_1 \models \langle I y \rangle (E(x, y) \& y > 20000) \& \neg M(x) \)

\[ \delta(\text{promote}(x), S_1) \models M(x) \]

**Engage** (x): \( \forall y, S_1 \models \langle I y \rangle E(x, y) \& \neg M(x) \)

\[ \delta(\text{engage}(x), S_1) \models M(x) \]

The **hire** (x) operation states that if it was always true in the past (the temporal assertion), and it is true in state \( S_1 \), that person x is not an employee, then in the state resulting from hiring x in state \( S_1 \), we will know that x is an employee with some salary \( y > 20000 \). Similarly, we may interpret the other operations.

### 4.9.5 Example: Analysis of an Insert Operation

Since we assume a finite universe of discourse, we can denote the elements of the universe of discourse as \( c_0, c_1, \ldots, c_n \). That is, in the rest of this section, we will use the following equivalences [trem75a]:

- \( (\forall x)P(x) \iff P(c_0) \& P(c_1) \& \ldots \& P(c_n) \)
- \( (\exists! x)P(x) \iff P(c_0) \lor P(c_1) \lor \ldots \lor P(c_n) \)
- \( (\exists x)^P(x) \iff P(c_0) \& \ldots \& \neg P(c_n) \)

Note that we use \( \exists! x \) to imply that the universe of discourse of \( s(x) \) is finite.

Similarly, the existential quantifiers which appear in the operation descriptions are replaced by \( \exists! \) in the following examples.

**EXAMPLE 1.** The verification of hire (x).

1. Applicability analysis:
   \[
   \neg \text{pre}(\text{hire}(x)) = \neg (\exists! y)E(x, y)
   \]

   \[
   \begin{array}{c|c|c|c}
   S_0 & S_1 & S_2 & S_3 \\
   \hline
   \neg E(a_0, b_0) & \neg E(a_0, b_0) & E(a_0, b_0) & E(a_0, b_0) \\
   b_0 > 20000 & b_0 > 20000 & b_0 > 20000 & b_0 > 20000 \\
   \neg M(a_0) & \neg M(a_0) & M(a_0) & M(a_0) \\
   E(a_0, 20000) & E(a_0, 20000) & \neg E(a_0, 20000) & \neg E(a_0, 20000) \\
   \hline
   (\exists! y)E(a_0, y) & (\exists! y)E(a_0, y) & (\exists! y)E(a_0, y) & (\exists! y)E(a_0, y) \\
   \hline
   E1 & E2 & E1 & E2 & E1 & E2 & E1 & E2 \\
   X & X & X & X & X & X & X & X \\
   \hline
   \end{array}
   \]

   applicable \quad \text{applicable} \quad \text{not applicable} \quad \text{not applicable}

   where \( E1 = E(x, 20000) \theta, E2 = E(x, b_0) \theta, \) and \( \theta = \{ \theta \} \).

   **Fig. 2.** Applicability analysis of the hire operation

   This analysis shows that the hire operation can only be applied in
state S0 and state S1 but not applicable in state S2 and state S3.

2. acceptability analysis:

\[ \text{post(hire}(\overline{x})) \]
\[ \neg (\exists y)(E(a_g, y) \land y \geq 20000) \]
\[ = (E(a_g, b_0) \land b_0 > 20000) \]
\[ \land \neg (E(a_g, 20000) \land b_0 > 20000) \]
\[ = (E(a_g, b_0) \land b_0 > 20000) \]
\[ \land \neg (E(a_g, 20000) \land \text{FALSE}) \]
\[ = (E(a_g, b_0) \land b_0 > 20000) \]

\[ S_0 \]
\[ \neg E(a_g, b_0) \]
\[ b_0 > 20000 \]
\[ \neg M(a_g) \]
\[ \neg E(a_g, 20000) \]
\[ S_1 \]
\[ \neg E(a_g, b_0) \]
\[ b_0 > 20000 \]
\[ \neg M(a_g) \]
\[ \neg E(a_g, 20000) \]
\[ S_2 \]
\[ E(a_g, b_0) \]
\[ b_0 > 20000 \]
\[ M(a_g) \]
\[ E(a_g, 20000) \]
\[ S_3 \]
\[ E(a_g, b_0) \]
\[ b_0 > 20000 \]
\[ M(a_g) \]
\[ E(a_g, 20000) \]

\[ E_1 \]
\[ X \]
\[ E_2 \]
\[ X \]
\[ E_3 \]
\[ X \]
\[ E_4 \]
\[ X \]
\[ E_5 \]
\[ X \]

not accepted not accepted accepted accepted

where \( E_1 = \neg E(a_g, b_0) \) and \( E_2 = (b_0 > 20000) \).

Fig. 10. Acceptability analysis of the hire operation

This indicates that after performing the hire(\( \overline{x} \)) operation, the system enters states 2 and 3.

3. Preservability analysis: For clarity, let \( P \) denote pre(hire(\( \overline{x} \))) and \( Q \) denote post(hire(\( \overline{x} \))) \[ \theta \], where \( \theta = \{ \langle a_g, \overline{x} \rangle \} \) (see applicability analysis of hire operation). Since the hire(\( \overline{x} \)) operation is applicable in state \( S_0 \), and accepted in states \( S_2 \) and \( S_3 \). We must check if

\[ S_0 = \{ \langle \alpha : P | \alpha \rangle \} \cap \{ \neg \alpha : Q | \alpha \} \cup \{ \alpha : Q | \alpha \} - \{ \alpha : P | \alpha \} \]

is equal to \( S_2 \) and \( S_3 \) respectively.

We calculate the "deleted statements" and the "added statements" first. The deleted statements are enclosed in the first pair of parentheses. The added statements are enclosed in the second pair of parentheses.

The set \( D \) of the deleted statements is

\[ D = \{ \alpha : P | \alpha \} \cap \{ \neg \alpha : Q | \alpha \} \]
\[ = \{ \alpha : \exists y \} E(a_g, y) | \alpha \} \cap \{ \neg \alpha : \exists y \} E(a_g, y) | y \geq 20000 | \alpha \} \]
\[ = \{ \neg E(a_g, b_0), E(a_g, 20000) \} \cap \{ \neg \alpha : E(a_g, b_0) | b_0 \geq 20000 | \alpha \} \]
\[ = \{ \neg E(a_g, b_0) \}

The set \( A \) of added statements is

\[ A = \{ \alpha : Q | \alpha \} \]
\[ = \{ \alpha : \exists y \} E(a_g, y) | y \geq 20000 | \alpha \} - \{ \alpha : \exists y \} E(a_g, y) | \alpha \} \]
\[ = \{ \alpha : E(a_g, b_0) | b_0 \geq 20000 | \alpha \} - \{ E(a_g, b_0) | E(a_g, 20000) \} \]
\[ = \{ E(a_g, b_0) | b_0 \geq 20000 \} - \{ E(a_g, b_0) | E(a_g, 20000) \} \]
\[ = \{ E(a_g, b_0) | b_0 > 20000 \} \]

Thus, \( S_0 = D \cup A \)

\[ S_0 - D \cup A \]
\[ = \{ E(a_g, b_0), b_0 > 20000, M(a_g), E(a_g, 20000) \} - \{ E(a_g, b_0) \} \cup A \]
\[ = \{ b_0 > 20000, M(a_g), E(a_g, 20000) \} \cup \{ E(a_g, b_0), b_0 > 20000 \} \]
which is not equal to $S_3$. This analysis indicates that the $\text{hire}(\bar{x})$ operation can be executed in $S_0$ and leads to $S_2$.

Thus, we have the state transition depicted below, where a question mark represents an illegal state.

\[ S_0 \xrightarrow{\text{hire}(\bar{x})} S_2 \]
\[ S_1 \xrightarrow{\text{hire}(\bar{x})} ? \]
\[ S_3 \]

5.9.6 Example: Analysis of a Delete Operation

EXAMPLE 2. The analysis of the $\text{fire}(\bar{x})$ operation.

1. Applicability analysis: (not applicable in $S_0$ and $S_1$, which is omitted).

\[ \text{pre}(\text{fire}(\bar{x})) \]
\[ = \neg((\exists y)((E(x,y) \& y > 20000) \& \neg M(\bar{x})) \]
\[ = E(x,b_0) \lor (b_0 > 20000) \lor M(\bar{x}) \]

| $S_2$ | $S_3$
|---|---|
| $E(a_0,b_0)$ | $E(a_0,b_0)$
| $b_0 > 20000$ | $b_0 > 20000$
| $\neg M(a_0)$ | $M(a_0)$
| $\neg E(a_0,20000)$ | $\neg E(a_0,20000)$

\[ E(x,b_0) \lor (b_0 > 20 k) \lor M(\bar{x}) \]

| applicable $\emptyset = \{<a_0,x>\}$ | not applicable

This shows that the $\text{fire}(\bar{x})$ operation is only applicable in state $S_2$, where substitution $\emptyset = \{<a_0,x>\}$.
2. Acceptability analysis:

\[ \neg \text{post(fire}(\bar{x})\text{)} \theta = (\exists y)E(\bar{x}, y) \]
\[ = E(\bar{x}, b_0) \lor E(\bar{x}, 20000) \]

\[
\begin{array}{cccc}
S_0 & S_1 & S_2 & S_3 \\
\neg E(a_0, b_0) & \neg E(a_0, b_0) & E(a_0, b_0) & E(a_0, b_0) \\
b_0 > 20000 & b_0 > 20000 & b_0 > 20000 & b_0 > 20000 \\
M(a_0) & M(a_0) & \neg M(a_0) & \neg M(a_0) \\
\neg E(a_0, 20000) & \neg E(a_0, 20000) & \neg E(a_0, 20000) & \neg E(a_0, 20000) \\
\end{array}
\]

\[
\begin{array}{llllll}
E_1 & E_2 & E_1 & E_2 & E_1 & E_2 \\
X & X & X & X & X & X \\
\end{array}
\]

applicable applicable not applicable not applicable

where \( E_1 = E(\bar{x}, b_0), \) \( E_2 = E(\bar{x}, 20000) \) and \( \theta = \{ a_0, \bar{x} \} \)

This indicates that the fire(\( \bar{x} \)) operation is accepted by state \( S_0 \) and \( S_1 \) respectively.

3. Preservability analysis:

1) The set \( D \) of deleted statements is

\[
D = \{ \alpha : P | \alpha \} \cap \{ \alpha : Q | \alpha \} \\
= \{ E(a_0, b_0), b_0 > 20000, \neg M(a_0) \} \cap \{ E(a_0, b_0), E(a_0, 20000) \} \\
= \{ E(a_0, b_0) \}
\]

The set \( A \) of added statements is

\[
A = \{ \alpha : Q | \alpha \} - \{ \alpha : P | \alpha \} \\
= \{ \neg E(a_0, b_0), \neg E(a_0, 20000) \} - \{ E(a_0, b_0), b_0 > 20000, \neg M(a_0) \} \\
= \{ \neg E(a_0, b_0), \neg E(a_0, 20000) \}
\]

Thus, \( S_2 = D \cup A \) is

\[
S_2 = D \cup A \\
= S_2 - \{ E(a_0, b_0) \} \cup A \\
= \{ b_0 > 20000, \neg M(a_0), \neg E(a_0, 20000) \} \cup \{ \neg E(a_0, b_0), \neg E(a_0, 20000) \} \\
= \{ \neg E(a_0, b_0), b_0 > 20000, \neg M(a_0), \neg E(a_0, 20000) \} \\
= S_0 \neq S_1
\]

This shows that fire(\( \bar{x} \)) goes from \( S_2 \) to \( S_0 \).

4.3.7 Analysis of Update Operations
4.9.1 Informal Introduction

We now discuss the analysis of 'update' operations.

For an 'update' operation, the problem is a little more complex. Unlike an 'insert' ('delete') operation, which changes the truth value of some ground prime from false to true (from true to false), an 'update' operation usually changes the values of some terms of the ground primes. For example, the updating operation $\text{raise}(x, 10\% \cdot y)$ increases the salary of $x$ by 10%. Using the method for analysing the applicability of the 'insert' and 'delete' operations, we see that the $\text{raise}(x, 10\% \cdot y)$ operation can be applied in states $S_2$ and $S_3$. However, if we use the same method to analyse the acceptability of the $\text{raise}(x, 10\% \cdot y)$ operation, we will find out that there exists no state which accepts the operation, where $\theta = \{ \langle a_0, x \rangle, \langle b_0, y \rangle \}$:

$$
\text{"post(raise}(x, 10\% \cdot y))\} = \text{"E}(a_0, b_0 + 10\% \cdot b_0)
$$

<table>
<thead>
<tr>
<th>$S_0$</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{E}(a_0, b_0)$</td>
<td>$\text{E}(a_0, b_0)$</td>
<td>$\text{E}(a_0, b_0)$</td>
<td>$\text{E}(a_0, b_0)$</td>
</tr>
<tr>
<td>$b_0 &gt; 20000$</td>
<td>$b_0 &gt; 20000$</td>
<td>$b_0 &gt; 20000$</td>
<td>$b_0 &gt; 20000$</td>
</tr>
<tr>
<td>$M(a_0)$</td>
<td>$M(a_0)$</td>
<td>$M(a_0)$</td>
<td>$M(a_0)$</td>
</tr>
<tr>
<td>$\text{E}(a_0, 20000)$</td>
<td>$\text{E}(a_0, 20000)$</td>
<td>$\text{E}(a_0, 20000)$</td>
<td>$\text{E}(a_0, 20000)$</td>
</tr>
<tr>
<td>not accepted</td>
<td>not accepted</td>
<td>not accepted</td>
<td>not accepted</td>
</tr>
</tbody>
</table>

**Fig. 11. Acceptability analysis of the raise operation**

Fig. 11 shows that the analysis of the acceptability of the raise operation cannot be performed by using the previous method. The reason is that in the post-state, the salary of $x = a_0$ has been changed. In order to show that $\text{E}(a_0, b_0 + 10\% \cdot b_0)$ is satisfied in some legal state, the legal state should contain the positive ground prime $\text{E}(a_0, b_0 + 10\% \cdot b_0)$. However, none of the four legal system states $S_0$, $S_1$, $S_2$, $S_3$ contains such a prime. In states $S_2$ and $S_3$, there is the positive prime $\text{E}(a_0, b_0)$. If we replace the salary of $a_0$ which is $b_0$ by $b_0 + 10\% \cdot b_0$, then the state resulting from such a replacement will be the post-state that we want. That is,

$$
\text{"post(raise}(x, 10\% \cdot y))\} = \text{"E}(a_0, b_0 + 10\% \cdot b_0)
$$

<table>
<thead>
<tr>
<th>$S_0$</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{E}(a_0, b_0)$</td>
<td>$\text{E}(a_0, b_0)$</td>
<td>$\text{E}(a_0, b_0)$</td>
<td>$\text{E}(a_0, b_0)$</td>
</tr>
<tr>
<td>$b_0 &gt; 20000$</td>
<td>$b_0 &gt; 20000$</td>
<td>$b_0 &gt; 20000$</td>
<td>$b_0 &gt; 20000$</td>
</tr>
<tr>
<td>$M(a_0)$</td>
<td>$M(a_0)$</td>
<td>$M(a_0)$</td>
<td>$M(a_0)$</td>
</tr>
<tr>
<td>$\text{E}(a_0, 20000)$</td>
<td>$\text{E}(a_0, 20000)$</td>
<td>$\text{E}(a_0, 20000)$</td>
<td>$\text{E}(a_0, 20000)$</td>
</tr>
<tr>
<td>not accepted</td>
<td>accepted if</td>
<td>accepted if</td>
<td>accepted if</td>
</tr>
</tbody>
</table>

**Fig. 12. Modified acceptability analysis of the raise operation**
4.9.7.2 Formal Analysis of Update Operations

The idea of substituting $b_0 + 10\%b_0$ for $b_0$ resembles the concept of unification as discussed at the beginning of this section. It indicates that in order to prove that the postcondition of an 'update' operation $\langle \text{op} \rangle$ is true in some legal system state $S_j$, we must try to find a substitution $\theta$ of the form

$$\theta = \{<t_1,v_1>, ..., <t_n,v_n> \}$$

where for $i = 1, ..., n$, $t_i$ is a term of $\text{post}(\langle \text{op} \rangle)$ and $v_i$ is an element of the parameter space of some sort such that $S_j \theta \cup \{\text{post}(\langle \text{op} \rangle)\}$ results in a closed tableau. That is, we prove that $S_j \theta \cup \{\text{post}(\langle \text{op} \rangle)\}$ is inconsistent.

For instance, the substitution for our example is

$$\theta = \{<x,a_0>, <\tilde{y}+10\%\tilde{y},b_0>\}$$. In order to prove that the operation is accepted in some legal system state $S_j$, we prove that

$$S_j \theta \cup \{\text{post}(\text{raise}*(x,10\%\tilde{y}))\}$$

has a closed tableau.

Similarly, since an 'update' operation changes only the values of some of the elements of some parameter spaces, which is represented by the substitution $\theta$, the preservability analysis of an 'update' operation should check if

$$S_i - \{\alpha: \text{pre}(\langle \text{op} \rangle)\theta|\alpha\} - \{\alpha: \text{post}(\langle \text{op} \rangle)\theta|\alpha\}$$

$$U (\{\alpha: \text{post}(\langle \text{op} \rangle)\theta|\alpha\} - \{\alpha: \text{pre}(\langle \text{op} \rangle)\theta|\alpha\})$$

$$= S_j \theta$$

where $S_i$ and $S_j$ denote the prestate and the poststate respectively. $S_j \theta$ denotes the state resulting from applying the substitutions $\theta$, $\theta$ consecutively to the legal system state $S_j$. The implication of the above criterion can be explained as follows. The set

$$\{\alpha: \text{pre}(\langle \text{op} \rangle)\theta|\alpha\}$$

is the set of ground instances of prime formulae that must be true when calling the procedure $\langle \text{op} \rangle$. The set

$$\{\alpha: \text{post}(\langle \text{op} \rangle)\theta|\alpha\}$$

denotes the set of ground instances of prime formulae that are expected to be true after executing the procedure. Thus, the difference between these two sets as enclosed in the first pair of parentheses denotes the "deleted statements". Similarly, the difference enclosed in the second pair of parentheses denotes the "added statements". Clearly, the prestate minus the deleted statements and then union the added statements must be identical to the poststate $S_j \theta$ for otherwise there will be some undesirable "side-effect" incurred by the transition from $S_i$ to $S_j \theta$. 
4.9.7.3 Example: Analysis of an Update Operation

EXAMPLE 3. The analysis of the raise* (x, 10% y).

1. Applicability analysis.

\[-pre(\text{raise}^*(x, 10\% y))\]
\[= \neg (E(x, y) \land y > 20000)\]
\[= E(x, y) \lor \neg (y > 20000)\]

<table>
<thead>
<tr>
<th>S₀</th>
<th>S₁</th>
<th>S₂</th>
<th>S₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>E2</td>
<td>E1</td>
<td>E2</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

not applicable  not applicable  applicable  applicable

\[E1 = E(x, y), \quad E2 = \neg (y > 20000) \quad \text{where } \theta = \{ \langle a₀, x \rangle, \langle b₀, y \rangle \}\]

2. Acceptability analysis: (not accepted by S₀ and S₁ which we omitted for clarity)

\[-post(\text{raise}^*(x, 10\% y))\]
\[= \neg (E(x, y + 10\% y) \lor \neg (y + 10\% y) > 20000)\]

<table>
<thead>
<tr>
<th>S₂</th>
<th>S₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>E(a₀', b₀')</td>
<td>E(a₀', b₀')</td>
</tr>
<tr>
<td>b₀' &gt; 20000</td>
<td>b₀' &gt; 20000</td>
</tr>
<tr>
<td>E(a₀', 20000)</td>
<td>E(a₀', 20000)</td>
</tr>
</tbody>
</table>

\[E(x, y + 10\% y)\]
\[\neg (y + 10\% y) > 20000\]

<table>
<thead>
<tr>
<th>S₂'</th>
<th>S₃'</th>
</tr>
</thead>
<tbody>
<tr>
<td>E(a₀', b₀')</td>
<td>E(a₀', b₀')</td>
</tr>
<tr>
<td>b₀' &gt; 20000</td>
<td>b₀' &gt; 20000</td>
</tr>
<tr>
<td>E(a₀', 20000)</td>
<td>E(a₀', 20000)</td>
</tr>
</tbody>
</table>

accepted  accepted

\[\text{where } \theta = \{ \langle x, a₀ \rangle, \langle y + 10\% y, b₀ \rangle \}\]

This indicates that the \text{raise}^*(x, 10\% y) operation is accepted by S₂ and S₃ respectively.

3. In a similar way, we can analyse the preservability of the example. We see that \text{raise}^*(x, 10\% y) goes from S₂ to S₂, and from S₃ to S₃. For simplicity, we will not carry out the analysis further here.

4.9.8 State Transition Diagram

After analysing all the operations we will have the state transition diagram below.
Fig. 13. The state transition diagram for the example

Fig. 13 gives us a state transition diagram which allows us to designate each of the states in turn as the current state and study the effect of a sequence of operations starting in this current state. Note that the transition from S2 to S3 (and from S3 to S3) has been labeled by raise*(x, 10% y)/θθ, to indicate that θ and θ should be consecutively applied to all the states that are reachable through the application of the raise* operation (see Sect. 4.9.7.2).

A transition which leads to the illegal state indicates that the application of the operation will yields illegal states, e.g., the engage(x) operation in Fig. 13. This means that we have to modify the operation description and/or the static constraints. It is beyond the scope of this thesis to discuss such modifications. For simplicity, we will ignore those transitions leading to the illegal state, this gives rise to the figure below.

Fig. 14. The transition diagram after removing the illegal transitions

4.9.9 Interpretation of The Transition Diagram

The interpretation of the state transition diagram should be made as follows. Suppose that in state S0 we hire John. That is, hire(John) can be applied in this state if S0 E (1y) E (John, y). After performing the operation, the system enters the state S2 which satisfies the postcondition of the operation. Now suppose we want to hire another individual Tom. In this case, the hire(Tom) operation cannot be executed iff the system is in state S0. A similar strategy is found in [80] for interpreting a conceptual model in REMORA.

Clearly, the second hire operation has no effect on the temporal constraint which says that
Whoever has been an employee cannot be hired again.

If, we want to consider the effect of hiring an individual on another individual, we must modify the operation descriptions so that the resulting state transition diagram has different states representing the different situations. For example, we may introduce a cardinality predicate which distinguishes several cases: 1) there exists no employee; 2) there is one employee; 3) there are more than one employee; etc. In this case, the fire operation is not allowed to be applied in a state in which no employee exists. In a state where more than one employee exists, then after firing an employee, the system may enter a state in which only one employee exists or enter a state in which there are more than one employee. A similar strategy is adopted by Furtado [35] where a dynamic modelling of an existence constraint is considered.

In the next section we will present how to incorporate the temporal assertions of the operations to transform the state transition diagram into a family of pushdown automata. We then present the consistency checking of the temporal constraints.

4.10 Verification of Temporal Constraints

The verification of temporal constraints is achieved in the following steps. First, we use the state transition diagram to form a family of n finite automata [48], where n is the number of states in the transition diagram. The finite automata are used for two purposes: it is used to construct a family of n pushdown automata by taking into account the temporal assertions of the operation descriptions. It is used to generate a collection of n sets of test sequences. A test sequence is a sequence of operations which can be deduced from a finite automaton. However, not all test sequences are applicable in the future information system because the history of the information system may not satisfy the temporal assertion of each operation in the test sequence.

To find out which test sequences are actually applicable in the future information system, we use the pushdown automata to analyse the test sequences. A test sequence is applicable in the future information system if it is accepted by some pushdown automaton constructed from the state transition diagram. For each test sequence that is accepted by some pushdown automaton, we check if it satisfies all the temporal constraints.

4.10.1 The Finite Automata

Corresponding to each state $S_i$ of the transition diagram in Fig.14, there is a finite automaton $fa_i(K, Q, \delta, S_i, F)$ which can be defined as follows (conf. [48]):

- $K$ is the set of states of the transition diagram.

- $Q$ is the set of operations labelling the transitions in the transition diagram, e.g., hire($x$), fire($x$), etc.

- $\delta : Q \times K \rightarrow K$ maps $<op_j>$, $S_k$ to $S_i$ iff there is a transition from $S_k$ to $S_i$ on $<op_j>$ in the transition diagram.
- $S_i$ is the **initial state** of $f_{a_i}$

- $F = K$ is the set of **final states** of $f_{a_i}$

Clearly, if the transition diagram has $n$ states $S_1, \ldots, S_n$, then there exist $n$ finite automata $f_{a_i}, i = 1, \ldots, n$. These finite automata differ only in the initial states. Each $f_{a_i}$ allows us to study the future behavior of the information system with respect to the current state $S_i$.

In fact, the $\delta$ mapping of the finite automata is exactly the $\delta$ construct that is used in describing the operations (see Sect.3.3.1).

### 4.10.2 The Pushdown Automata

Corresponding to each $f_{a_i}(K, \Omega, \delta, S_i, F)$ there is a pushdown automaton $p_{a_i}(K, \Omega, \Gamma, \delta', S_i, Z, F)$, where the states, the operations, the initial state and the final states are the same for $p_{a_i}$ and $f_{a_i}$. The other elements of $p_{a_i}$ are defined as follows:

- $\Gamma$ is a finite alphabet called the **pushdown alphabet**.
  - $\Gamma$ contains any temporal assertions including aux.

- $Z$ in $\Gamma$ is a particular temporal assertion which implies every temporal assertion. $Z$ initially appears on the pushdown store.

- $\delta'$ is a mapping from $\Omega \times (K \times \Gamma)$ to $K \times \Gamma$, which is to be defined below.

The definition of the $\delta'$ (delta') mapping will be carried out in Sections 4.10.2.1, 4.10.2.2, 4.10.2.3 and 4.10.2.4.

### 4.10.2.1 Some Temporal Axioms

We define some temporal axioms which will be used to define the $\delta'$ mapping. Let $w_1, w_2$ be two temporal assertions to the past excluding the present. We have

- **A1:** $\text{exi} * w_1 \leftrightarrow \text{almax} * (\neg w_1)$
- **A2:** $\text{almax} * w_1 \rightarrow \text{exi} * w_1$
- **A3:** $\text{almax} * (w_1 \land w_2) \leftrightarrow \text{almax} * w_1 \land \text{almax} * w_2$
- **A4:** $\text{exi} * (w_1 \lor w_2) \leftrightarrow \text{exi} * w_1 \lor \text{exi} * w_2$
- **A5:** $\text{almax} * w_1 \lor \text{almax} * w_2 \rightarrow \text{almax} * (w_1 \lor w_2)$
- **A6:** $\text{exi} * (w_1 \land w_2) \rightarrow \text{exi} * w_1 \land \text{exi} * w_2$
- **A7:** $\text{almax} * w_1 \rightarrow w_1$
- **A8:** $w_1 \rightarrow \text{exi} * w_1$
- **A9:** $\neg \text{exi} * w_1 \rightarrow \neg w_1$
- A10: \( Z \rightarrow w_1 \)
- A11: \( w_1 \rightarrow \text{any} \)

\( w_1 \) and \( w_2 \) are inconsistent iff \( w_1 \rightarrow \neg w_2 \) or \( w_2 \rightarrow \neg w_1 \). Otherwise, \( w_1 \) and \( w_2 \) are said to be consistent.

4.10.2.2 Informal Presentation of the Lambda Mapping

This section introduces the \( \lambda \) (lambda) mapping which will be used in the definition of the \( \delta' \). Simply speaking, \( \lambda \) maps a pair

\[
\langle \text{w}_{\text{history}}, \text{w}_{\text{just-happened}} \rangle
\]

of temporal assertions to a temporal assertion \( \text{w}_{\text{new-history}} \). \text{w}_{\text{history}} is the historic recording of the information system at an abstract level. That is, it denotes the history of the system by means of a temporal assertion. \text{w}_{\text{history}} changes when an operation has been executed. \text{w}_{\text{just-happened}} is a temporal assertion of the form

\[
\text{ever}^+ \text{pre}(\langle \text{op} \rangle)
\]

where \( \langle \text{op} \rangle \) is the operation that has been executed. Clearly, the fact that \( \langle \text{op} \rangle \) has been executed implies that once in the past the precondition of the operation \( \langle \text{op} \rangle \) must have been true. Hence,

\[
\text{ever}^+ \text{pre}(\langle \text{op} \rangle)
\]

is intended to denote this fact, which is to be combined with the history of the system by the \( \lambda \) mapping to obtain a new history \( \text{w}_{\text{new-history}} \) of the system.

In defining the \( \lambda \) mapping, several cases should be considered:

1) If the existing history already implies \( \text{w}_{\text{just-happened}} \), then the new system history will be the existing one.

2) If the existing history is in conflict with \( \text{w}_{\text{just-happened}} \), then the new history should be

"It is not the case that \text{always} in the past the existing history was true and it is not the case that \text{always} in the past what happened was true."

That is, the new history should be

\[
\neg \text{always}^+ \text{w}_{\text{history}} \& \neg \text{always}^+ \text{w}_{\text{just-happened}}
\]

3) If the existing history does not imply \( \text{w}_{\text{just-happened}} \) nor that the existing history is in conflict with \( \text{w}_{\text{just-happened}} \), then it is enough to "remember" that something has happened, i.e., the new history should be

\[
\text{w}_{\text{history}} \& \text{w}_{\text{just-happened}}
\]
4.10.2.3 Formal Definition of the Lambda Mapping

We may now formally define the $\lambda$ mapping as follows. Let $w_1$ be any temporal assertion to the past and $w_2$ be a temporal assertion of the form

$$\forall x \in P \phi$$

where $P$ denotes the precondition of some operation in question and $\phi$ denotes the substitution with which the operation can be applied in some state. We have

- $\lambda(w_1, w_2) = w_1$ if $w_1 \rightarrow w_2$

- $\lambda(w_1, w_2) = \neg\text{always}^+ w_1 \land \neg\text{always}^+ w_2$ if $w_1$ and $w_2$ are inconsistent.

- $\lambda(w_1, w_2) = w_1 \land w_2$ if neither $w_1 \rightarrow w_2$ nor $w_1$ and $w_2$ are inconsistent.

4.10.2.4 The Delta' Mapping

Let $w$ denote the temporal assertion and $P$ the precondition of an operation in question. For any state $S_k$ in $K$ which is the current state of $p_{ai}$, any operation $<op>$ in $Q$ that is being considered, and any temporal assertion $w'$ in $\Gamma$ which appears as the top element of the pushdown store:

- Case 1 in which $w' \in Z$:

$$\delta'(S_k, <op>, w') = <S_1, \lambda(w' \forall x \in P \phi)>$$

where $S_1 = \delta(<op>, S_k)$ is determined by the finite automaton $fa_i$; that is, the legal state resulting from applying $<op>$ in $S_k$. If no such $S_1$ can be found, the mapping is undefined.

- Case 2 in which $w' \in \Gamma - \{Z\}$:

$$\delta'(S_k, <op>, w') = <S_1, \lambda(w' \forall x \in P \phi)>$$

if $w'$ implies $w\theta$;

otherwise $\delta'(S_k, <op>, w')$ is undefined, where $S_1 = \delta(<op>, S_k)$, which is determined from the finite automaton $fa_i$.

4.10.2.5 The Behavior of the Pushdown Automata

Intuitively, $p_{ai}$ acts as follows (see the figure below). In a system state $S_1$, stack top $w'$, and an operation $<op>$, the pushdown automaton $p_{ai}$ checks if $w'$ satisfies the temporal assertion $w\theta$ of $<op>$ when it is to be applied in some state. That is, if $w' = w\theta$. If so, the machine makes a move; it goes to another state $S_1$ according to the finite automaton $fa_i$, and tries to "remember" that something new has happened. That is, it combines the history of the "database" (i.e., the stack top $w'$) with the new information (i.e., $\forall x \in P \phi$) according to the $\lambda$ mapping and push the result, i.e., $\lambda(w' \forall x \in P \phi)$, onto the stack. Thus, the stack top always denotes the database history at an abstract level.
Since we are interested in knowing the effect of a sequence of operations on an arbitrary state, we designate one of the state as the initial state of the pushdown automaton $p_{a_i}$, i.e., $S_1$. We start the $p_{a_i}$ with $Z$ on the stack. This means that nothing has been remembered. A sequence of operations can be accepted iff starting the $p_{a_i}$ with such a configuration one can finish the sequence. A sequence of operations is rejected iff it is not accepted by any $p_{a_i}$ for all $i$.

4.10.3 Generate Test Sequences

In this section, we present a method for generating the test sequences which are used to verify the consistency of the temporal constraints. The method is adapted from [25].

4.10.3.1 The Testing Tree

Given a finite automaton $f_{a_i}$, a testing tree $T$ is constructed as follows:

1) Label the root of $T$ with the initial state of $f_{a_i}$. This is level 0 of $T$.

2) Suppose we have already built $T$ to a level $j$. The $(j+1)$th level is built by examining nodes in the $k$th level from left to right. A node at the $j$th level is terminated if its label is the same as a nonterminal at some level $h \leq k$. Otherwise, let $S_k$ denote its label. If on input $<op>$, $f_{a_i}$ goes from state $S_k$ to state $S_1$, we attach a branch and a successor node to the node labeled $S_k$ in $T$. The branch and the successor node are labeled with $<op>$ and $S_1$ respectively.

3) Attach to each node $S_k$ of the tree all the operations that are applicable in state $S_k$.

The above process always terminates, since there are only a finite number of states in $f_{a_i}$. Also, depending on the order in which we place the successor node, a different tree may result.

EXAMPLE. Suppose that we have designated $S_0$ of our finite automaton obtained earlier as the initial state (see Fig.14). We have the following testing tree.
4.10.3 Test Sequences

A partial path of the testing tree is a sequence of consecutive branches. It starts at the root and ends at either a terminal or nonterminal node, say $S_i$. For each partial path, there are a number of test sequences obtained by concatenating the partial path with each of the operations which are attached to the node at which the partial path ends. In the rest of the thesis, $\tau_i$ denotes the set of all test sequences generated by using $S_i$ as the initial state.

For instance, our example has the following $\tau_i$'s. For simplicity, we denote hire by $h$, fire by $f$, raise/88 by $r/88$ and promote by $p$.

$$\tau_0 = \{ h, f; h, r/88; h, p; h, f, h; h, r/88, f; h, r/88, r/88; h, r/88, p; h, p, r/88; h, p, r/88, r/88 \};$$

$$\tau_2 = \{ r/88, r/88; r/88, f; r/88, p; f, h; f, h, r/88; f, h, f; f, h, p; p, r/88; p, r/88, r/88 \};$$

$$\tau_3 = \{ r/88, r/88 \}.$$

4.10.4 Acceptance of Test Sequences

For each test sequence $\langle op_1 \rangle \ldots \langle op_n \rangle$ in $\tau_i$, we analyse if it can be accepted by the pushdown automaton $pa_i$. A sequence of operations $\langle op_1 \rangle \ldots \langle op_n \rangle$ is accepted by $pa_i$ iff $pa_i$ starts in $S_i$ with $Z$ on the top of the stack, $\langle op_1 \rangle$ as the input, and makes a sequence of moves which finishes the sequence $\langle op_1 \rangle \ldots \langle op_n \rangle$.

4.10.4.1 Example: the Rejection of a Test Sequence

We show the method by analysing the test sequence $h, f, h, C, g$, i.e., the sequence of operations hire, fire, hire. For the descriptions of these operations, see Sect. 4.9.4. The figure below illustrates this test.
We cannot carry on because the stack top does not imply the temporal assertion of the hire operation. That is, the test sequence is rejected by the pushdown automaton $\mathcal{P}_0$.

### 4.10.4.2 Example: The Acceptance of a Test Sequence

Another example analyses the test sequence hire, promote, raise*/88.

<table>
<thead>
<tr>
<th>current</th>
<th>store top</th>
<th>〈op›</th>
<th>next state</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_0$</td>
<td>$Z$</td>
<td>hire</td>
<td>$S_2$</td>
</tr>
<tr>
<td>$S_2$</td>
<td>$\lambda (a \lambda x x x^+ \langle \lambda y E(a_0, y) \rangle, x x x^+ \langle \lambda y E(a_0, y) \rangle) = a \lambda x x x^+ \langle \lambda y E(a_0, y) \rangle$</td>
<td>fire</td>
<td>$S_0$</td>
</tr>
<tr>
<td>$S_0$</td>
<td>$\lambda (a \lambda x x x^+ \langle \lambda y E(a_0, y) \rangle, x x x^+ \langle \lambda y E(a_0, y) \rangle, - M(a_0)) = x x x^+ \langle \lambda y E(a_0, y) \rangle, x x x^+ \langle \lambda y E(a_0, y) \rangle, - M(a_0)$</td>
<td>hire</td>
<td></td>
</tr>
</tbody>
</table>

This example shows that the test sequence hire, promote, raise*/88 is accepted by $\mathcal{P}_0$.

### 4.10.5 Execution Sequences

The examples in the previous sections indicates that only a subset of $\tau_1$ is accepted by $\mathcal{P}_1$, where $\tau_1$ is the set of test sequences generated by using $S_1$ as the initial state. For each element $\langle \text{op}_1 \rangle \ldots \langle \text{op}_m \rangle$ of the subset which is accepted by $\mathcal{P}_1$, there is a sequence of states

$$S_1 S_1 j \ldots S_{j-1}$$

such that $S_{j+1}$ is entered from $S_{j+1}$ on $\langle \text{op}_j \rangle$ in $\mathcal{P}_1$. We call such a sequence of states a partial execution sequence, denoted as $\alpha_p$. It is partial because it represents only the future seen from $S_1$ or only the past seen from $S_{j-1}$ but not both. For instance, the test sequence hire, promote, raise*/88 corresponds to the execution sequence

$$S_0 S_2 S_3 S_3 88$$
where $S_0S_1S_2S_3S_4 S_5$ represents the future resulting from applying the test sequence in $S_0$. On the other hand, $S_0S_2S_3$ represents the past leading to $S_3S_4 S_5$ by some sequence of operations, which is applied in some state. To form the set of total execution sequences, we first take the union of all the subsets of $\tau_i$ which is accepted by $p_{a_i}$ for $i = 1, \ldots, n$, where $n$ is the number of states of the transition diagram. This union is the set of all the partial execution sequences which is denoted as $\Sigma_p$. The set of total execution sequences $\Sigma$ is formed as follows. If $\sigma_1S_k$ and $\sigma_1S_0$ is in $\Sigma_p$, then $\sigma_1S_kS_0$ is in $\Sigma$, where $S_k$ is interpreted as the current state of the information system, $\sigma_1$ is interpreted as the history and $S_0$ is interpreted as the future of the information system. In order to distinguish the name of the state from the relative position of the state with respect to the current state, we denote an execution sequence in the following form

$$\ldots k1k2k3k4k5 \ldots$$

$$\ldots -2 -1 0 1 2 \ldots$$

where $k1, \ldots, k5$ are the names of the states and $-2, -1, 0, 1, 2$ are the relative positions of the states with respect to the current state. In particular, $S_k$ is the current state because its position (or "offset") relative to the current state is 0.

### 4.10.6 Verification of the Temporal Constraints

For each element

$$\sigma = \ldots k1k2k3 \ldots$$

in $\Sigma$ we check if $\sigma$ satisfies all the temporal constraints. This check can be done by using the semantics which was defined in Sect.3.2.3.

As an example, we show that

$$\sigma = S_0S_2S_0S_2S_0S_0$$

$$\ldots -2 -1 0 1 2 \ldots$$

satisfies the temporal constraint

$$tc_1: \text{ Whoever has been an employee cannot be hired.}$$

$$( \text{exec+} (I)E(x,y) \rightarrow \text{EXECUTABLE}(\text{hire}(x)) )$$

It is proved by contradiction as follows. We negate $\sigma \not\vdash tc_1$ and infer a contradiction, which implies that $tc_1$ must be satisfied by $\sigma$.

$$\neg(\sigma \not\vdash tc_1)$$

$$\leftrightarrow \neg (\sigma \not\vdash (\text{exec+} (I)E(x,y) \rightarrow \text{EXECUTABLE}(\text{hire}(x))) )$$

$$\leftrightarrow \neg (\sigma \not\vdash (\text{exec+} (I)E(x,y)) \land \text{EXECUTABLE}(\text{hire}(x)))$$

$$\leftrightarrow \neg (\exists x \forall y (I)E(x,y) \land \text{EXECUTABLE}(\text{hire}(x)))$$

That is, we have to prove that not both $\exists x \forall y I E(x,y)$ and $\text{EXECUTABLE}(\text{hire}(x))$ can be true at the same time. We see that $\exists x \forall y I E(x,y)$ is true iff
Case 1: $S^0 \vdash (Iy)E(x,y)$ or
Case 2: $S^2 \vdash (Iy)E(x,y)$ or
Case 3: $S^0 \vdash (Iy)E(x,y)$

However, Case 1 and Case 3 are not true since state $S^0$ does not satisfy $(Iy)E(x,y)$. This is equivalent to that $(Iy)E(x,y)$ is true iff Case 2 is true. However, Case 2 is true only when $a_0$ is substituted for $x$. Thus it suffices to prove that $\not\exists$EXECUTABLE(hire(x)) is not true when $x$ is substituted by $a_0$.

Recalling that $\not\exists$EXECUTABLE(⟨op⟩) if (see Sect.3.2.3):

a) $\not\exists$the temporal assertion of the operation ⟨op⟩.

b) $\not\exists$the precondition of the operation ⟨op⟩. and

c) $\not\exists$the postcondition of the operation ⟨op⟩.

We have

\[
[ \not\exists$EXECUTABLE(hire(x)) ]
\]

\[<==> \not\exists \text{always} \uparrow^n (Iy)E(a_0,y) \quad \{\text{condition a}\}
\]
\[& \not\exists (Iy)E(a_0,y) \quad \{\text{condition b}\}
\]
\[& \delta(hire(x),S^0) \vdash (Iy)E(a_0,y) \quad \{\text{condition c}\}
\]

\[<==> (\forall y \in 0)(a_0 \not\exists (Iy)E(a_0,y))
\]
\[& \not\exists (Iy)E(a_0,y) \quad \{\text{condition a}\}
\]
\[& \delta(hire(x),S^0) \vdash (Iy)E(a_0,y) \quad \{\text{condition c}\}
\]

\[<==> S^0 \vdash a_0 \not\exists (Iy)E(a_0,y)
\]
\[& \not\exists (Iy)E(a_0,y) \quad \{\text{condition b}\}
\]
\[& \delta(hire(x),S^0) \vdash (Iy)E(a_0,y) \quad \{\text{condition c}\}
\]

\[<==> S^2 \vdash (Iy)E(a_0,y)
\]
\[& S^2 \vdash (Iy)E(a_0,y) \quad \{\text{condition b}\}
\]
\[& \delta(hire(x),S^0) \vdash (Iy)E(a_0,y) \quad \{\text{condition c}\}
\]

\[<==> FALSE
\]

since $S^2 \vdash (Iy)E(a_0,y)$ is not true. This indicates that the execution sequence fulfills $tC_1$.

4.11 Summary Discussions

In this chapter, we have presented the modified tableau approach for testing the consistency of a conceptual model.

We want to point out the following points:

1) It has been proved in [104] that for a temporal language which deals with only the future (respectively only the past) including the present, it suffices to consider test sequences of length at most $n$, where $n$ is the number of states of the transition diagram. This justifies the method that is used for generating the test sequences.
2) The modified tableaux approach has been experimented by Mow to the specification and verification of communication protocols, where a prolog program for verification has been implemented [49].

3) It is pointed out in [49] that the construction of the system states uses exponential computation time. The reason is that the problem of truth assignments for a set C of clauses over a set V of statement variables is NP-complete. The problem of constructing one system state from the resulting tableau tree in Sect.4.6 can be rephrased as the problem below [38]:

**INSTANCE:** V is a set of statement variables, C is a collection of clauses over V.

**QUESTION:** Is there a satisfying truth assignment for C?

**REFERENCE:** [27]. It is referred to as the Cook's Theorem.

**Comment:** Given a set V of statement variables and a collection C of clauses over V, it is required to find out an assignment of truth values to the statements in V such that every clause in C is true.

Let \( \text{c} \in C \) be a clause. The number of primes in \( \text{c} \) is denoted by \( |c| \). For example, if \( \text{c} = \text{E}(x, 20000) \text{ V M(a)} \), then \( |c| = 2 \). In [38], it is pointed out that the problem of finding out a satisfying truth assignment for \( C \) remains NP-complete even if each \( c \) in \( C \) satisfies \( |c| = 3 \), or if each \( c \) in \( C \) satisfies \( |c| < 3 \) and, for each \( v \) in \( V \), there are at most 3 clauses in \( C \) that contain either \( v \) or \( \neg v \). The general problem is solvable in polynomial time if each \( c \) in \( C \) has \( |c| < 2 \) [31].

The problem of constructing the (legal) system states for a set of wffs is NP-complete since we can regard the set of ground primes as the set V of statement variables and the set of the associate sets as the collection C of clauses over V. Thus, the construction of a legal system state for the set of wffs is equivalent to finding out whether there is a satisfying truth assignment for the set of associate sets.

4) Another problem that is associated with the modified tableaux approach is that the restriction that is imposed upon the static constraints is too strong. For instance, the formula

\[ (\forall x)(\exists y)\text{WF}(x, y) \]

is consistent however it is excluded from consideration by the modified tableaux approach. In the next chapter, we will present an improved approach for testing the consistency of a specification, which imposes the weakest restriction upon the static constraints. That is, further reduction of the restriction will result in a class whose consistency is undecidable [59].
CHAPTER 5:

CONSISTENCY CHECKING:

AN IMPROVED APPROACH
5 CONSISTENCY CHECKING: AN IMPROVED APPROACH

In the last chapter, we have described the modified tableaux approach for testing the consistency of information systems specifications. As pointed out in the last chapter, the modified tableaux approach has two weaknesses:

1) It imposes a restriction upon the static constraints, which is too strong in most cases.

2) Its computation time is exponential because the problem of constructing the system states is NP-complete.

In this chapter, we will present an improved approach for consistency checking. This approach is suggested by the works of Lewis [59]. The approach also consists of three components: consistency checking of the static constraints, operation analysis and the consistency checking of the temporal constraints. However, the methods that are used to test the static constraints and to analyse the operations are different from the methods that are used in the last chapter. The verification of the temporal constraints is the same as in the last chapter. Therefore, we will not discuss this aspect in this chapter.

The layout of this chapter is as follows:

- Sect.5.1 presents the work of Lewis, which shows that compact sets of clauses are solvable. However, the result of Lewis is presented from a theoretical point of view which cannot be efficiently applied in practice.

- Sect.5.2 presents our definition of compactness which is more suitable to be used for our purposes. Our definition of compactness is presented in terms of a directed graph called the unifiability digraph. We prove, in this section, that our definition is equivalent to the one defined by Lewis.

- Sect.5.3 presents some results concerning the unifiability digraph defined in Sect.5.2.

- Sect.5.4 through Sect.5.6 present some examples which illustrate the verification of the static constraints.

- Sect.5.7 presents the improved method for operation analysis.

5.1 Lewis' Work

As pointed out in Sect.4.3, the consistency of a set W of wffs is in general undecidable. However, if certain restrictions are imposed upon the set W of wffs, then the consistency of W can be effectively determined. One possible restriction has been discussed in the last chapter, where recursiveness among the quantifiers is excluded. It should be pointed out that such a restriction is in general too strong. That is, it excludes many interest classes of wffs from consideration. Moreover, the method suffers from NP-completeness. This means that the number of unit operations increases exponentially as the degrees of the wffs increase.
A weaker condition for restricting a set \( W \) of \( \text{wffs} \) is found in [59]. This condition is defined in terms of the \textit{clause form} of \( W \). A \( \text{wff} \) is said to be in clause form if it is of the form

\[
\alpha_1 \lor \alpha_2 \lor \ldots \lor \alpha_n
\]

where \( \alpha_i, i=1,\ldots,n, \) is an atomic formula or the negation of an atomic formula. \( \alpha_i \) is also called a \textit{prime formula} or a \textit{literal}. A method for transforming a \( \text{wff} \) into its clause form is included in Appendix A. The concept of unification is also used in the following definition. For the definition of unifier and the unification algorithm, see Appendix B.

\textbf{DEFINITION 5.1.} Let \( S \) be a set of clauses. An \textit{s-link} is an ordered triple \( \langle C, \alpha, \beta \rangle \), where \( C \) is a clause in \( S \) and \( \alpha, \beta \) are distinct literals in \( C \). An \textit{s-chain} is a sequence \( \langle C_1, \alpha_1, \beta_1 \rangle, \ldots, \langle C_n, \alpha_n, \beta_n \rangle \) of \( s \)-links such that for \( i=1, \ldots, n-1, \beta_i \) is unifiable with \( \alpha_{i+1} \); this \( s \)-chain is said to have \textit{length} \( n \). If \( \beta_n \) is unifiable with \( \alpha_1 \), then this \( s \)-chain is called an \textit{s-cycle}. [59]

For example, let \( S \) consist of the following set of clauses

\[
\begin{align*}
\neg E(x_1) & \lor WF(x_1, f(x_1)) \\
\neg E(x_2) & \lor s(x_2) < 20000 \\
WF(x_3, y_3) & \lor M(y_3) \\
M(y_4) & \lor \neg s(y_4) < 20000
\end{align*}
\]

According to Definition 5.1, \( S \) has the following 8 \( s \)-links, since the example has 4 clauses each of which has two literals.

\[
\begin{align*}
\langle C_1, \neg E(x_1), WF(x_1, f(x_1)) \rangle \\
\langle C_1, WF(x_1, f(x_1)), \neg E(x_1) \rangle \\
\langle C_2, \neg E(x_2), s(x_2) < 20000 \rangle \\
\langle C_2, s(x_2) < 20000, \neg E(x_2) \rangle \\
\langle C_3, WF(x_3, y_3), M(y_3) \rangle \\
\langle C_3, M(y_3), WF(x_3, y_3) \rangle \\
\langle C_4, \neg M(y_4), s(y_4) < 20000 \rangle \\
\langle C_4, s(y_4) < 20000, \neg M(y_4) \rangle
\end{align*}
\]

One of the longest \( s \)-chains of this example is of length 4 and is shown below.

\[
\begin{align*}
\langle C_1, \neg E(x_1), WF(x_1, f(x_1)) \rangle \\
\langle C_3, WF(x_3, y_3), M(y_3) \rangle \\
\langle C_4, \neg M(y_4), s(y_4) < 20000 \rangle \\
\langle C_2, s(x_2) < 20000, \neg E(x_2) \rangle
\end{align*}
\]

Clearly, if we start from \( C_2 \), then we may have another longest \( s \)-chain. It can be easily verified that the example contains no \( s \)-cycle.

\textbf{DEFINITION 5.2.} A set \( S \) of clauses is \textit{compact} if \( S \) contains no \( s \)-cycle.[59]

\textbf{THEOREM 5.1.} \textit{Satisfiability} is decidable for compact sets of clauses.

For proof of Theorem 5.1, see [59]. Theorem 5.1 means that if a set \( S \) of clauses is compact, then it can be determined within a finite number of steps whether \( S \) is consistent or inconsistent. In other
words, if $S$ is compact, then only finite many new clauses can be produced by unrestricted resolution. It is proved in [59] that compact sets of clauses are the maximal sets for which the consistency is decidable. That is, further reduction of the restriction will result in classes of formulae whose consistency is undecidable.

We propose a definition of compactness in the following section and then we prove that our definition are equivalent to Lewis'.

5.2 Our Definition of Compactness

In this section, we will define the concept of compactness in terms of a directed graph. Some terminology concerning directed graphs are clarified as follows.

5.2.1 Terminology for Directed Graphs

A directed graph or digraph $G$ is an ordered pair $\langle V, E \rangle$, where $V$ is a finite set of points and $E \subseteq V \times V$ is a finite set of arcs. $\langle u, v \rangle$ is allowed to be in $E$. Arc $\langle u, v \rangle$ goes from $u$ to $v$.

The outdegree of a point is the number of arcs going from it. The indegree of a point is the number of arcs coming into it.

A path is a sequence $v_0, \ldots, v_n$ of points $(n \geq 0)$ such that $\langle v_i, v_{i+1} \rangle$ is an arc for $i = 0, \ldots, n-1$, such that the $v_i$'s are distinct. Except that we allow $n \geq 0$ and $v_0 = v_n$, in which case the path is a cycle. A digraph with no cycles is acyclic. If there is a path from $u$ to $v$, then $v$ is reachable from $u$. $u$ is reachable from $v$ for all $u$.

Semipath and semicycle are defined like path and cycle, except that either $\langle v_i, v_{i+1} \rangle$ or $\langle v_{i+1}, v_i \rangle$ may be an arc.

A digraph is weakly connected if there is a semipath between every pair of points. A digraph which is not weakly connected consists of weak components each of which is maximally weakly connected. A digraph is strongly connected if every two points are mutually reachable.

5.2.2 Unifiability Digraph

DEFINITION 5.3. A unifiability digraph $G(S) = \langle S, E \rangle$ for a set $S$ of clauses is defined as follows.

1) $G(S) = \langle S, E \rangle$ has $S$ as its points and $E \subseteq S \times S$ as its set of arcs.

2) $\langle C_i, C_j \rangle$ belongs to $E$ if for some positive $\alpha \in C_i$, negative $\beta \in C_j$ such that $\alpha$ is unifiable with $\beta$ for some unifier $\theta$. In this case, we label the arc by $C_i \cdot \alpha/C_j \cdot \beta \cdot \theta$. (Note that $i$ and $j$ may be equal.)

Two remarks are made here. First, we allow more than one arc going from $C_i$ to $C_j$. However, these arcs are labeled differently. Second, an empty substitution may be a unifier, this may happen when two ground atomic formulae are in conflict with each other.
**Definition 5.4.** Let \( G(S) \) be a unifiability digraph of \( S \). Two arcs
\[
e_1 = C_{i_1} \cdot \alpha_1 / C_{j_1} \cdot \beta_1 + \theta_1
\]
and
\[
e_2 = C_{i_2} \cdot \alpha_2 / C_{j_2} \cdot \beta_2 + \theta_2
\]
belonging to \( G(S) \) are said to be conflicting if \( C_{i_1} = C_{i_2} \) and \( \alpha_1 = \alpha_2 \) or \( C_{j_1} = C_{j_2} \) and \( \beta_1 = \beta_2 \). E.Q.

In the first case we say that \( C_{i_1} \) has \( e_1 \) and \( e_2 \) as a pair of its conflicting arcs; in the second case we say that \( C_{j_1} \) has \( e_1 \) and \( e_2 \) as its conflicting arcs.

In practice, we may drop the designators \( C_{i_1}, C_{i_2}, C_{j_1}, C_{j_2} \) because these designators can be easily determined from a diagram of the graph.

**Definition 5.5.** A semicycle of \( G(S) \) is said to be conflicting if it contains a point \( C \) together with a pair of its conflicting arcs. E.Q.

Note that by Definition 5.4, a cycle of \( G(S) \) is never conflicting.

**Definition 5.6.** A unifiability digraph \( G(S) \) is said to be compact if \( G(S) \) contains only conflicting semicycles if any. E.Q.

For example, let \( S \) be the set of clauses:
\[
\begin{align*}
C_1 & : \neg WF(x_1, a) \lor WF(b, x_1) \\
C_2 & : \neg WF(x_2, x_2) \lor \neg WF(y_2, x_2) \\
C_3 & : WF(b, a)
\end{align*}
\]
then \( G(S) \) will be

<table>
<thead>
<tr>
<th>G(S)</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>( WF(b, a) / \neg WF(y_2, x_2) )</td>
<td>( C_3 ) is indegree zero and ( C_2 ) is outdegree zero. However we can determine immediately that ( G(S) ) is compact since ( C_3 ) is a conflicting point as well as ( C_1 ) and ( C_2 ). ( C_3 ) is a conflicting point because ( WF(b, a) = WF(b, a) ). A similar argument can be made for ( C_1 ) and ( C_2 ).</td>
</tr>
</tbody>
</table>

---

**5.2.3 Proof of a Theorem**

**Lemma 5.1.** If \( G(S) \) is compact, then \( S \) is compact.

Proof. Suppose that \( G(S) \) is compact and there is an \( S \)-cycle. Let \( \langle C_1, \alpha_1, \beta_1 \rangle, \ldots, \langle C_n, \alpha_n, \beta_n \rangle \) be the shortest \( S \)-cycle. Then all the \( C_i \)'s are distinct for \( i = 1, \ldots, n \). If \( C_i = C_j \) for some \( i < j \) then \( \langle C_{i+1}, \alpha_{i+1}, \beta_{i+1} \rangle, \ldots, \langle C_{j-1}, \alpha_{j-1}, \beta_{j-1} \rangle, \langle C_j, \alpha_j, \beta_j \rangle \) would be a shorter \( S \)-cycle.
If \( n=1 \) then \( G(S) \) has a loop which is not conflicting, violating our assumption that \( G(S) \) is compact. (Recall that \( G(S) \) is compact if it contains only conflicting semicycles, if any).

If \( n=2 \) then since \( \alpha_1 \) and \( \beta_1 \) are distinct and \( \alpha_2 \) and \( \beta_2 \) are distinct, which follows from the definition of \( S \)-links. Now if \( \alpha_1, \beta_1 \) are positive then \( C_2 \) is entered by \( \alpha_1/\beta_2 \) and \( \beta_1/\alpha_2 \). This reveals that there is a non-conflicting semicycle. Similarly, if \( \alpha_1, \beta_1 \) are negative, the \( C_1 \) is entered by \( \alpha_2/\beta_1 \) and \( \beta_2/\alpha_1 \) which implies a non-conflicting semicycle. On the other hand, if \( \alpha_1, \beta_1 \) have different signs, then there is a cycle which is also not conflicting.

If \( n>2 \), then

- If every \( \alpha_i \) is positive and \( \beta_i \) is negative (or vice versa), then there is a cycle in \( G(S) \) which is not conflicting. Or

- If some \( S \)-link in the \( S \)-cycle, say \( \langle C_i, \alpha_i, \beta_i \rangle \) has positive literals \( \alpha_i \), \( \beta_i \) (or negative literals \( \alpha_i \) and \( \beta_i \)), then a semicycle which corresponds to the \( S \)-cycle is shown as follows:

\[
\begin{array}{cccccccc}
C_1 & \ldots & C_{i-1} & C_i & \ldots & C_{n} \\
\ \downarrow & & \ \downarrow & & \ \downarrow \\
& & & & & & & \\
& e_{i-1} & \alpha_1 & \beta_1 & e_i & \beta_{n-1} & \alpha_n \\
& & & & & & & \\
& (\beta_1/\alpha_2) & & & (\beta_{n-1}/\alpha_n) & & & \\
\end{array}
\]

where \( e_{i-1} = C_i \cdot \alpha_i / C_{i-1} \cdot \beta_{i-1} \) and \( e_i = C_i \cdot \beta_i / C_{i+1} \cdot \alpha_{i+1} \). Since \( C_1, \ldots, C_n \) are all distinct, so are \( C_{i-1} \) and \( C_{i+1} \). Moreover, \( \alpha_i \) and \( \beta_i \) are distinct because \( \langle C_i, \alpha_i, \beta_i \rangle \) is an \( S \)-link, which requires that \( \alpha_i \) and \( \beta_i \) must be distinct. If \( \alpha_i = \beta_i \) then \( \langle C_i, \alpha_i, \beta_i \rangle \) would not have been an \( S \)-link and hence \( \langle C_1, \alpha_1, \beta_1 \rangle, \ldots, \langle C_n, \alpha_n, \beta_n \rangle \) would not have been an \( S \)-cycle. Thus the semicycle above is non-conflicting. This contradicts the assumption that \( G(S) \) is compact. E.Q.

**Lemma 5.2.** If \( S \) is compact then \( G(S) \) has only conflicting semicycles if any.

**Proof.** Suppose that \( G(S) \) has a semicycle which is not conflicting. Let \( C_1, C_2, \ldots, C_n, C_1 \) denote the shortest semicycle so that \( C_1, \ldots, C_n \) are all distinct. If \( C_1, \ldots, C_n, C_1 \) is a cycle then there is an \( S \)-cycle so \( S \) is not compact.

We assume that \( C_1, \ldots, C_n, C_1 \) is not a cycle.

If \( n=1 \) then \( G(S) \) has a loop which is also a cycle. This case has been excluded above.

If \( n=2 \) then since \( C_1, C_2, C_1 \) is not a cycle, we must have either
Since $C_1, C_2, C_3$ is not conflicting, i.e., $\alpha_1 \neq \beta_1$ and $\alpha_2 \neq \beta_2$, we must have $\langle C_1, \alpha_1, \beta_1 \rangle, \langle C_2, \alpha_2, \beta_2 \rangle$ being an $S$-cycle.

If $n > 2$ then since $C_1, \ldots, C_n, C_1$ is not a cycle, we must have either

\[
\begin{array}{c}
\alpha_2 / \beta_1 \\
C_1 \rightarrow C_2 \\
(\beta_2 / \alpha_1)
\end{array}
\]

or

\[
\begin{array}{c}
\alpha_1 / \beta_n \\
C_1 \rightarrow C_2 \\
(\beta_1 / \alpha_i)(\beta_i / \alpha_{i+1})
\end{array}
\]

where $\alpha_i \neq \beta_i$ and $\beta_{i-1} \neq \alpha_{i+1}$, otherwise $C_1, \ldots, C_n, C_1$ would have been conflicting.

In either case we must have an $S$-cycle $\langle C_1, \alpha_1, \beta_1 \rangle, \ldots, \langle C_{n-1}, \alpha_n \rangle$. E.Q.

**Theorem 5.2.** If $G(S)$ is compact then the satisfiability of $S$ is decidable.

Proof. This theorem follows from Lemma 5.1, Lemma 5.2 and Theorem 5.1. E.Q.

### 5.3 Some Results for $G(S)$

In this section, we present some results for $G(S)$. Some of these results can be used to determine the consistency of $S$ immediately. Some of the results enable us to reduce the effort that is required to determine the consistency of $S$.

The following Lemma enables us to consider the weak components separately.

**Lemma 5.3.** Suppose that $G(S)$ is not weakly connected, and let $\langle S_1, E_1 \rangle$, $\langle S_2, E_2 \rangle$, $\ldots$, $\langle S_n, E_n \rangle$ be the weak components of $G(S)$. Then $S$ is satisfiable iff each of $S_1, \ldots, S_n$ is satisfiable.

A proof of Lemma 5.3 can be found in [59] where it is given for a different unifiability digraph. However, the proof can be equally applied to $G(S)$. 


By virtue of Lemma 5.3, we may from now on consider only weakly connected digraphs.

In the proof of the next Lemma, we will use the concept of Herbrand universe $H$ for a set $S$ of clauses, which is recursively defined as follows:

1) Let all the constant symbols mentioned in $S$ be in $H$. If there is no constant symbol in $S$, then we allow some arbitrary constant symbol to be in $H$.

2) Suppose that $t_1, \ldots, t_m$ are in $H$. Then $F(t_1, \ldots, t_m)$ is in $H$, where $F$ is any function symbol mentioned in $S$.

3) No other elements except those defined in 1) and 2) are in $H$.

If $P$ is a set of terms and $S$ is a set of clauses, then $P(S)$ is the set of all ground instances of clauses from $S$ resulting from substituting terms from $P$ for variables in $S$. In particular, if $H$ is the Herbrand universe of $S$, then $H(S)$ is the set of all ground clauses resulting from substituting terms from $H$ for variables in $S$.

It can be shown that $S$ is satisfiable (or consistent) iff $H(S)$ is satisfiable. Further, $S$ is satisfiable iff $P(S)$ is satisfiable for every finite subset $P$ of $H$.

By $\mathbb{U}H(S)$ we mean the set of all literals which appear in $H(S)$.

Following [59], a model for a set of ground clauses is a set of ground literals which contains no ground literal together with its negation and, which has non-empty intersection with each of the clauses.

The following Lemma can be used to determine if a set of clauses is satisfiable.

**Lemma 5.4.** If $G(S)$ contains no point of indegree zero, or no point of outdegree zero, then $S$ is satisfiable.

Proof. In the first case each clause contains a negative literal, in the second case each clause contains a positive literal. If $H$ is the Herbrand universe of $S$ and $M$ is the set of all atomic formulae appearing in $H(S)$, then $\{ \neg A : A \in M \}$ is a model for $H(S)$ in the first case and $M$ is a model for $H(S)$ in the second case. That is, $H(S)$ is satisfiable and hence $S$ is satisfiable.[59] E.Q.

**Corollary.** If $G(S)$ is strongly connected then $S$ is satisfiable. In particular, if $G(S)$ consists of only one point, then $S$ is satisfiable.

We prove the following Lemma.

**Lemma 5.5.** If $G(S)$ consists of only a single, non-conflicting semicycle, then $S$ is consistent.

Proof. Let $C_1, C_2, \ldots, C_n, C_1$ be the non-conflicting semicycle. By Lemma 5.2, there is an $S$-cycle. This $S$-cycle includes every clause of $S$ for otherwise there would have been another (shorter) semicycle violating the assumption of the lemma. Let $\langle C_1, \alpha_1, B_1 \rangle, \ldots, \langle C_n, \alpha_n, B_n \rangle$ denote this $S$-cycle. If $H$ is the Herbrand universe of $S$, then $H(\{ \alpha_1, \ldots, \alpha_n \})$ is a model of $S$. E.Q.
A point \( C \) of \( G(S) \) is called a **redundant point** if \( C \) contains a literal \( \alpha \) which is not unifiable with the negation of any other literals in \( S \). \( G(S) \) is called redundant if it contains a redundant point.

**Lemma 5.6.** \( S \) is consistent iff \( S - \{ C_i \} \) is consistent, where \( C_i \) is a redundant point of \( G(S) \).

**Proof.** If \( S \) is consistent then \( S - \{ C_i \} \) is consistent since every subset of a consistent set is also consistent.

Let \( H \) be the Herbrand universe of \( S \). Let \( \alpha_i \in C_i \) be the literal that is not unifiable with the negation of any other literal. Suppose that \( S - \{ C_i \} \) is consistent, then \( H(S - \{ C_i \}) \) is also consistent. Let \( M_i \) be a model of \( H(S - \{ C_i \}) \). Then \( M_i \cup H(\alpha_i) \) is a model for \( H(S) \) since \( \alpha_i \) is not unifiable with the negation of any other literals in \( S \). This implies that \( S \) is consistent since \( H(S) \) is consistent. \( \Box \).

Lemma 5.6 is also called **Robinson's purity principle**. It is very useful in reducing a set \( S \) of clauses to a subset of \( S \) in practice. We will see some examples which make use of this Lemma. By virtue of Lemma 5.6, we may sometimes consider only **non-redundant** unifiability digraphs.

### 5.4 Example 1

Let \( S \) consist of the following sentences

1. Every employee works for someone.
   
   \( (\forall x)(\exists y)( E(x) \rightarrow WF(x,y) \) )

2. WF(x,y) implies that y is a manager.
   
   \( (\forall x)(\forall y)( WF(x,y) \rightarrow M(y) \) )

3. Every manager is an employee.
   
   \( (\forall x)( M(x) \rightarrow E(x) \) )

The clause form of the example is

\[
\begin{align*}
C_1 & : \neg E(x_1) \lor WF(x_1,f(x_1)) \\
C_2 & : \neg WF(x_2,y_2) \lor M(y_2) \\
C_3 & : \neg M(x_3) \lor E(x_3) \\
\end{align*}
\]

We note that \( WF(x_1,f(x_1)) \) in \( C_1 \) is unifiable with \( \neg WF(x_2,y_2) \) in \( C_2 \), so we can draw an arc from \( C_1 \) to \( C_2 \) which is labeled by \( WF(x_1,f(x_1))/\neg WF(x_2,y_2) \). \( M(y_2) \) in \( C_2 \) is unifiable with \( \neg M(x_3) \) in \( C_3 \) which implies an arc from \( C_2 \) to \( C_3 \). Similarly there is an arc from \( C_2 \) to \( C_1 \). The resulting digraph is depicted in the following figure (we omit the substitutions).
\[
\begin{align*}
E(x_3)/\neg E(x_1) \\
C_1 \rightarrow C_2 \rightarrow C_3 \\
WF(x_1,f(x_1)) \quad M(y_2)/\neg M(x_3) \\
/\neg WF(x_2,y_2)
\end{align*}
\]

**Fig. 15. The digraph for the set \{ C_1, C_2, C_3 \}**

We see that the indegrees of C_1, C_2 and C_3 are all 1, this means that the digraph contains no point of indegree zero, therefore \{ C_1, C_2, C_3 \} is satisfiable (or consistent) according to Lemma 5.4. Note that the universe of discourse or the Herbrand universe is infinite, this infinity is induced by the recursiveness of the three formulae.

It is important to point out that the above example makes no use of the concepts of S-links, S-chains and S-cycles. The satsifiability or consistency can be determined merely by examining the digraph of the example.

### 5.5 Example 2

The set of static constraints below is borrowed from [64], the meanings of the predicate symbols are:

- **S(x):** x is a secretary.
- **E(x):** x is an employee.
- **P(x,y):** x is paid with y.
- **W(x):** x is a wage.

\[
\begin{align*}
sc_1: & \quad \text{Every secretary is an employee.} \\
(\forall x)(S(x) \rightarrow E(x)) \\
sc_2: & \quad \text{Every employee is paid with some wage.} \\
(\forall x)(\exists y)(E(x) \rightarrow P(x,y)) \\
sc_3: & \quad \text{P(x,y) implies that x is an employee and y is a wage.} \\
(\forall x)(\forall y)(P(x,y) \rightarrow E(x) \land W(y)) \\
sc_4: & \quad \text{Every employee has only one wage.} \\
(\forall x)(\forall y)(\forall z)(P(x,y) \land P(x,z) \rightarrow y = z)
\end{align*}
\]

The identity axioms are defined as follows:

- **A1:** The reflexive axiom. \\
  \[(\forall x)(x = x)\]
- **A2:** The symmetric axiom. \\
  \[(\forall x)(\forall y)(x = y \rightarrow y = x)\]
- **A3:** The transitive axiom. \\
  \[(\forall x)(\forall y)(\forall z)(x = y \land y = z \rightarrow x = z)\]

The set of wffs \{ sc_1, sc_2, sc_3, sc_4, A1, A2, A3 \} can be converted to the following set of clauses:
The digraph $G(S)$ can be shown as below. For clarity, we adopt the convention of labelling the arcs: 1) The literals of clause $C_i$ is numbered by the order in which they occur in $C_i$. For example, the two literals of $C_1$ is numbered as 1.1 and 1.2, respectively. That is, 1.1 denotes $\neg S(x_1)$ and 1.2 denotes $E(x_1)$, respectively. 2) If there are several arcs going from point $i$ to point $j$ in $G(S)$, then we will only draw one arc but label this arc with several labels separated by commas. This convention should be clear from the figure below.

![Diagram](image)

**Fig. 16. The unifiability digraph for $\{C_1, \ldots, C_8\}$**

By Lemma 5.6 we can remove points $C_1$ and $C_4$ from the above digraph (Fig.16) since $\neg S(x_1)$ of $C_1$ and $W(y_4)$ of $C_4$ are not unifiable with the negations of any other literals. We see that the outdegrees of all of the points after removing $C_1$ and $C_4$ are non-zero. By Lemma 5.4, we know immediately that $\{C_2, C_3, C_5, C_6, C_7, C_8\}$ is consistent and hence $\{C_1, \ldots, C_8\}$ is consistent as well.

### 5.6 Example 3

The example below is borrowed from [68], which contains the following clauses:

$C_1:\ \neg F(x,z) \lor \neg F(z,y) \lor G(x,y)$

The father of a father is a grandfather.

$C_2:\ \neg B(x,y) \lor \neg B(z,y) \lor B(x,y)$
The brother of a brother is a brother.

\[ C_3: \sim F(z,x) \lor \sim F(z,y) \lor x=y \lor B(x,y) \]
Two children of the same father are brothers.

\[ C_4: \sim M(x,y) \lor S(x,f(x,y),\text{male}) \]
In any marriage tuple, the husband is a male.

\[ C_5: \sim M(x,y) \lor S(y,g(x,y),\text{female}) \]
In any marriage tuple, the wife is a female.

\[ C_6: \sim S(x,y,z) \lor A(y) \]
The second argument of the predicate \( S \) is an age.

\[ C_7: \sim M(x,y) \lor \sim L(x,z) \lor L(y,z) \]
The wife lives in the same town as her husband.

\[ C_8: \sim L(x,y) \lor \sim L(x,z) \lor y=z \]
Anybody lives in only one town.

\[ C_9: \sim S(x,y,z) \lor y<150 \]
An age is less than 150.

\[ C_{10}: x=x \]
The reflexive axiom for identity.

\[ C_{11}: \sim (x=y) \lor y=x \]
The symmetric axiom for identity.

\[ C_{12}: \sim (x=y) \lor \sim (y=z) \lor x=z \]
The transitive axiom for identity.

\[ C_{13}: \sim (x<x) \]
The irreflexive axiom of \( < \).

\[ C_{14}: \sim (x<y) \lor \sim (y<x) \]
The asymmetric axiom of \( < \).

\[ C_{15}: \sim (x<y) \lor \sim (y<z) \lor x<z \]
The transitive axiom of \( < \).

For simplicity, we assume that there is no connection between the identity and the \( < \) relation. The unifiability digraph is shown on the next page.
Fig. 17. The unifiability digraph for Example 3

We see that the digraph in Fig. 17 consists of three weak components:

\[ \mathcal{G}(\{C_1\}) = \langle \{C_1\}, \emptyset \rangle \]

\[ \mathcal{G}(\{C_2, C_3, C_7, C_8, C_{10}, C_{11}, C_{12}\}) \]

\[ \mathcal{G}(\{C_4, C_5, C_6, C_9, C_{13}, C_{14}, C_{15}\}) \]

as depicted in Fig. 17. By Lemma 5.3, we may consider the these weak components separately. \( \{ C_1 \} \) is consistent. \( \{ C_2, C_3, C_7, C_8, C_{10}, C_{11}, C_{12} \} \) is also consistent since its digraph contains no point of outdegree zero and by Lemma 5.4, it is consistent.

We now consider \( \mathcal{G}(\{C_4, C_5, C_6, C_9, C_{13}, C_{14}, C_{15}\}) \). We see that point \( C_6 \) is redundant since \( A(y) \) of \( C_6 \) does not connect to any other point. This can be easily discovered by the observation that \( C_6 \) contains a positive literal \( A(y) \) however there is no arc going out from \( C_6 \). In a similar way, we see that \( C_4 \) and \( C_5 \) are also redundant. By virtue of Lemma 5.6, we may remove \( C_4, C_5 \) and \( C_6 \) from the digraph. The result is shown below:
We see that point $C_9$ becomes redundant and hence can be removed for the digraph. After removing $C_9$ we have

We see that there is no point of indegree zero and by Lemma 5.4 we can conclude that \{ $C_{13}$, $C_{14}$, $C_{15}$ \} is consistent. This in turn shows that \{ $C_4$, $C_5$, $C_6$, $C_9$, $C_{13}$, $C_{14}$, $C_{15}$ \} is consistent.

Since each of the weak components is consistent we conclude that \{ $C_1$, $C_4$, ..., $C_{15}$ \} is consistent by virtue of Lemma 5.3.

These examples show that it is sometimes possible to determine the consistency of the static constraints directly from the unifiability digraph. In information systems field, this is very useful since our intention is to have a consistent set of static constraints. Using a theorem-prover, we usually can prove the inconsistency of the specification but not the consistency of the specification. The value of the method presented in these sections is that it is possible to determine the consistency which is expected from a specification.

Example 3 above indicates that more information could have been associated with the digraph so that the process of the digraph would become more efficient. For instance, we could have associated with each point the number of literals it contains. When we find out that

\[
\{ x : (I_{ij}.y) \mid i.x/j.y \text{ is an outgoing arc of point } C_i \} \\
\cup \{ x : (I_{ij}.y) \mid i.x/j.y \text{ is an incoming arc of point } C_i \} \\
< \mid C_i \mid
\]

then point $C_i$ can be removed by virtue of Lemma 5.6, where $\mid C_i \mid$ denotes the number of literals in $C_i$. In this thesis, we will not pursue such implementation issues.
Although it is sometimes possible to determine the consistency directly from the digraph, it is not always possible to do so. For example, the consistency (or inconsistency) of the set of clauses below cannot be determined by the method presented in these two sections:

\[ C_1: \neg WF(x, \text{Chief-manager}) \lor WF(Tom, x) \]
Tom works for anybody who works for the Chief-manager.

\[ C_2: \neg WF(x, x) \lor \neg WF(y, x) \]
Nobody works for some body who works for himself.

\[ C_3: WF(Tom, \text{Chief-manager}) \]
Tom works for the Chief-manager.

The unifiability digraph of this example is shown below.

```
  1.2/2.2, 1.2/2.1  
  |                  
  v                  
  3.1/1.1, 3.1/2.2  
  |                  
  v                  
  C_3                
```

We see that the digraph is not redundant, and there is a point of indegree zero (i.e., \( C_2 \)) and a point of outdegree zero (\( C_3 \)). The consistency or inconsistency of the set cannot be determined directly from the method that has been presented. In this case, we have to use a theorem-prover, e.g., the resolution principle [78]. Specifically, if the unifiability digraph of the set of clauses is compact, then only finite many new clauses can be produced by unrestricted resolution [59]. Thus, when the empty clause is produced by the theorem-prover or when there is no new clause can be generated by the theorem-prover, the theorem-prover is stopped. In the former case, the set is inconsistent and in the latter case, the set is consistent.

In Appendix C, we include a method for testing the consistency or inconsistency of a set of clauses. The method is based upon the works by Prawitz [75] and Robinson [78], respectively.

### 5.7 Operation Analysis in the Improved Method

As discussed in the last few sections, there is a procedure for testing if a compact set of clauses is consistent. Moreover, we know that any set of static constraints can be transformed into a set of clauses. Therefore we may assume in this section that the consistency of the static constraints has been formally checked.

#### 5.7.1 Informal Discussion of Basic Ideas

For easy understanding, we discuss the basic ideas in terms of a relational database.

Suppose that we have a relational database with only one static constraint:

\[ sc1': (\forall x)(\exists y)(M(x) \rightarrow E(x, y) \land y > 20000) \]

which states that every manager is an employee having some salary more than 20000. For simplicity, we assume that there are only one person identified by name n and two salary values \( s < 20000 \) and \( s' > 20000 \).
It can be easily seen that $S1$, $S2$, $S3$ in Fig. 20a are three legal database states since $sc1'$ is true in each of them. Now consider the operation $f(n)$, i.e., fire the person identified by name $n$, which is applicable in a legal database state if $\langle n,s'\rangle \notin E$ and after applying $f(n)$, $\langle n,s'\rangle$ is deleted from $E$. That is, the precondition for applying the operation is $\langle n,s'\rangle \notin E$ and the postcondition of the operation is $\langle n,s'\rangle \notin E$.

Depending on the state in which $f(n)$ is applied, different state transitions may occur [22]. There are three transitions to be considered in this example which we denote as $f1$, $f2$ and $f3$ for easy explanation (Fig. 20b).

\[
\begin{align*}
M & \quad E \\
S1 & \{\} \quad \{\langle n,s'\rangle\} \\
S2 & \{\} \quad \{} \\
S3 & \{\langle n \rangle\} \quad \{\langle n,s'\rangle\} \\
S4 & \{\langle n \rangle\} \quad \{} \\
S5 & \{\} \quad \{\langle n,s\rangle\} \\
S6 & \{\langle n \rangle\} \quad \{\langle n,s\rangle\}
\end{align*}
\]

![Fig. 20. A transition diagram for fire and promote](image)

Transition $f1$ has the following properties:

1) The operation is applicable in a legal database state $S1$.

2) The application of the operation yields some legal database state $S2$.

3) Anything that is not specified to be changed by the operation remains true in the resulting state (i.e., the frame problem).

Transition $f2$ does not have the last property since it has deleted $\langle n \rangle$ from $M$ which is not specified in the $f(n)$ operation. Transition $f3$ does not have the second property and hence it should never occur in a database. Transition $f3$ is not a desirable transition. However, which of the transitions $f1$ and $f2$ is the desirable (or intended) transition cannot be determined from the operation description. In other words, we may say that the operation description is not sufficient for carrying out the operation for the following reasons:

If transition $f1$ is wanted, then the operation description should contain in the precondition that the operation can be applied if $\langle n,s'\rangle \notin E$ and $\langle n \rangle \in M$.

If transition $f2$ is wanted, then the operation description should contain in the postcondition that after applying the operation, $\langle n,s'\rangle \notin E$ and $\langle n \rangle \notin M$.

Suppose that the relational database imposes one more static constraint

\[sc2': \quad (\forall x)(\forall y)( E(x,y) \rightarrow y < 20000 ) \]
which states that every employee has salary less than 20000. sc1' and sc2' are consistent since S2 and S5 in Fig.20a are two legal database states, although in this case S1 and S3 are no longer legal. Now consider the operation p(n), i.e., promote n, which specifies that if \( <n,s> \in E \) and \( <n,s'> \in M \) then its application leads to the state in which \( <n,s'> \in E \), i.e., p1 in Fig.20b. However S6 is not a legal database state since sc1' does not hold.

If we change the postcondition so that after applying the operation, \( <n,s> \in M \) and \( <n,s'> \) is replaced by \( <n,s'> \), i.e., p2 in Fig.20b, then sc2' will not hold. In fact, there is no way to promote any individual to be a manager because sc1' and sc2' together prevent us from inserting any tuple into M.

In the following discussion, we denote the precondition of an operation by P and the postcondition of an operation by Q.

The basic steps for analysing the operation descriptions in the improved approach can be outlined below:

I) First we analyse the insert and delete operations:

   i) For each insert or delete operation \( <op> \) we check if there is a legal system state in which the precondition P is true (i.e., applicability analysis [35]). We then characterize the prestate in which P is true by a set \( \Psi \) of literals such that any legal system state must contain \( \Psi \) as a subset.

   ii) Next, we analyse whether the application of the operation yields some legal system states. In this test, we also consider the frame problem so that the resulting poststate contains the set \( \Psi' \) of literals, which are true in the prestate and not specified to be falsified by the operation. Clearly, \( \Psi' \subseteq \Psi \). Furthermore, the characterization of the poststate also contains the literals that are logical consequences of the postcondition and \( \Psi' \).

   iii) The two tests above give us a transition from the prestate to the poststate caused by the application of the operation in question. This transition can be depicted as:

   ![Fig. 21. A transition from Si to Sj on <op>](image)

   where \( S_i \) and \( S_j \) are the unique names given to the prestate and the poststate of the operation \( <op> \).

   iv) After analysing all the insert and delete operations, we will have a list of state transitions each of which consists of a prestate, a poststate and a transition denoted by the operation (Fig.21).
v) We then compare the prestates and the poststates and merge some of them to form a state transition diagram. Each state of the transition diagram can be seen as an abstract system state. Further, each of these states is legal.

II) Second, we analyse the update operations:

i) An update operation is applicable iff there is a state $S_i$ of the transition diagram and a substitution $\theta$ of terms $t_k$ of $S_i$ for parameters $X_k$ of the precondition $P$ of the update operation such that $P\theta$ is true in $S_i$. As usual, we denote $\theta$ by $\theta(\langle t_1, X_1 \rangle, \ldots, \langle t_n, X_n \rangle)$ where $k \neq l$ implies $X_k \neq X_l$ for $k, l = 1, \ldots, n$. $P\theta$ denotes the result of substituting the $t_k$'s for the $X_k$'s throughout $P$. Recall that parameters are allowed to range over all the values of the information systems states, $P\theta$ is then the instantiation of the precondition in the system state $S_i$.

ii) After applying the update operation, some attribute values of some relations of the "database" have been modified in the way as specified in the postcondition $Q$. The modification can be represented by an "extended substitution" $\tilde{\theta} = \{ \langle t'_1, t_1 \rangle, \ldots, \langle t'_n, t_n \rangle \}$ where $t'_k$, $t_k$ are terms of $Q$ and the "database" state being updated respectively and $k \neq l$ implies $t'_k \neq t'_l$ for $k, l = 1, \ldots, n$. Clearly, the application of the update operation yields some legal system state iff there is a state $S_j$ of the transition diagram and an extended substitution $\tilde{\theta}$ such that $S_j \tilde{\theta}$ satisfies $Q$. Usually, there may be more than one such state as we have shown in Chapter 4.

iii) To identify the desirable state $S_j$ we take into account the frame problem again. That is we require that those literals that are not specified to be changed by the update operation should be true in $S_j \tilde{\theta}$. The analysis of all the update operations will add some transitions into the transition diagram obtained in the last step. However, the update transitions will be labeled by $\langle \text{op} \rangle / \theta \theta$ which means that the substitutions $\tilde{\theta}$, $\theta$ should be consecutively applied to all the states reachable through the transition $\langle \text{op} \rangle / \theta \theta$. This will be further explained in the next subsection.

5.7.2 Proofs of Two Theorems

First note that when we start to analyse the operations, we already have a consistent set $SC$ of static constraints. Second, following Gallaire and Nicolas and Minker [36] [68] [67], we distinguish two kinds of static constraints: those that are used as "integrity constraints" and those that are used as derivation rules. For example, we do not (and cannot) explicitly store all the tuples of the "$<$" relation in an information system. Instead, the tuples are defined by some algorithm or derivation rules (e.g., $\langle x < x \rangle, x < y \& \& y < z \rightarrow x < z, x < y \rightarrow (y < x)\rangle$. Whenever needed, we can always derive any pair $\langle x, y \rangle$ which satisfies the $<$ relation. On the other hand, the static constraint stating that all managers are employees should be regarded as an integrity constraint. This implies that in a legal system state $S_i$, if $\langle n > EM$ then $\langle n, s > CE$ must be true.
Let \( P \) and \( Q \) denote the precondition and the postcondition of an operation in question. Without loss of generality, we may assume that \( P \) and \( Q \) are in clause form. Let \( SC \) and \( SC_C \subseteq SC \) denote the set of static constraints and integrity constraints respectively. Further, let \( C_{SC} \) denote the clause form of \( SC \) and \( C_I \) denote the clause form of \( SC_I \). We have the following

**Theorem 5.3.** An operation is applicable in some legal system state iff \( P \cup C_{SC} \) is consistent.

Proof. Suppose that \( P \) is true in some legal system state \( S_i \). That is, \( S_i \models P \). Since \( S_i \) is a legal system state, hence \( S_i \models C_{SC} \). Therefore \( S_i \models (P \cup C_{SC}) \). By virtue of the extended completeness theorem, \( P \cup C_{SC} \) must be consistent.

Now suppose that \( P \cup C_{SC} \) is consistent and let \( S_j \) be a (mathematical) model of \( P \cup C_{SC} \). We have \( S_j \models C_{SC} \) and hence \( S_j \) is a legal system state. Since \( S_j \models P \) and hence the operation in question is applicable in some legal system state. E.Q.

Let \( C \) be any set of clauses. Let \( \Pi(C) \) denote the union of \( C \) with the set of all clauses that can be obtained by resolution between pairs of clauses in \( C \). By \( \Pi^k(C) \) we mean \( \Pi(\Pi^{k-1}(C)) \), and \( \Pi^0(C) = C \). It can be proved that for all \( c \) for all \( k \), \( c \in \Pi^k(C) \) implies that \( c \) is a logical consequence of \( C \). In particular, the set of literals logically following \( C \) is defined by

\[ \Psi(C) = \{ \alpha : \alpha \text{ is a literal } \& \ (\exists k) (\alpha \in \Pi^k(C)) \} \]

We have

**Theorem 5.4.** Let \( S_i \) be a legal system state and \( \alpha \) any literal. If \( \alpha \in \Pi^k(C_I \cup P) \) (resp. \( \Pi^k(C_I \cup Q) \)) for some \( k \) and \( S_i \models P \) (resp., \( S_i \models Q \)), then \( S_i \models \alpha \).

Proof. We prove the case for the precondition. Suppose that \( S_i \) is a legal system state and \( S_i \models P \). Since \( S_i \) is legal, i.e., \( S_i \models C_{SC} \).

Since \( SC_I \subseteq SC \) which implies that \( C_I \subseteq C_{SC} \) and hence \( S_i \models C_I \). Thus, \( S_i \models (P \cup C_I) \).

For every \( \alpha \), \( \alpha \in \Pi^k(C_I \cup P) \) implies that \( \alpha \) is a logical consequence of \( C_I \cup P \). This implies that every model of \( C_I \cup P \) is a model of \( \alpha \). Since \( S_i \models (C_I \cup P) \) implies that \( S_i \) is model of \( C_I \cup P \) and hence \( S_i \) is also a model of \( \alpha \). That is, \( S_i \models \alpha \). E.Q.

**Corollary.** If \( C_I \cup P \) (resp. \( C_I \cup Q \)) is compact, then \( \Psi(C_I \cup P) \) (resp. \( \Psi(C_I \cup Q) \)) is finite and hence \( S_i \) is finite.

### 5.7.3 Formal Analysis of Insert and Delete Operations

If \( \langle \text{op} \rangle \) denotes an insert or delete operation and \( P \) and \( Q \) denote the precondition and the postcondition of \( \langle \text{op} \rangle \), then

1) \( \langle \text{op} \rangle \) is applicable in some legal system state iff
\( C \subseteq U P \)

is consistent (Theorem 5.3). If so, the prestate of \(<op>\) is characterized by the set (Theorem 5.4)

\[
\text{prestate}(<op>) = \psi(C \cup UP)
\]

This set is not a complete representation of the prestate but only a characterization of a relevant part of a legal system state such that the integrity constraints are true in it. The characterization of the prestate is similar to the use of the integrity constraints as generation rules under a closed world assumption [conf. [68]].

If \( C \cup UP \) is compact, then \( \psi(C \cup UP) \) is finite since only finite many new clauses can be produced by unrestricted resolution. If \( C \cup UP \) is not compact, then we have to remove some of the \( SC \) and use them as derivation rules in order to guarantee that \( S \) is finite (Theorem 4.2).

ii) The application of \(<op>\) yields some legal system state iff \( \text{poststate}(<op>) \cup C \subseteq \) is consistent, where

\[
\text{poststate}(<op>) = \psi(Q \cup \psi(<op>)) \cup \psi(<op>)
\]

and \( \psi(<op>) = \{ \alpha : \alpha \in \psi(\text{PU}C) \} \cup \{ \alpha \} \cup Q \) is consistent

Intuitively, \( \psi(<op>) \) denotes those literals that are true in the prestate and are not falsified by (the postcondition of) \(<op>\).

iii) The operation description is sufficient for characterizing the poststate if

\[
\psi(C \cup UQ) \subseteq \text{poststate}(<op>). \quad \text{(Theorem 4.2)}
\]

iv) If one of the above test fails, we must modify the static constraints and/or the operation description and repeat the whole process until each operation description passes the above tests.

v) At this stage, we should have a list of state transitions each of which can be depicted as

\[
\begin{array}{c}
S_i \quad \text{prestate}(<op_i>) \\
\hline
\text{<op_i>} \\
\hline
S_j \quad \text{poststate}(<op_i>)
\end{array}
\]

where \( S_i, S_j \) are the (unique) names given to the prestate and the poststate of the operation \(<op_i>\).

vi) We now merge some of the prestates and/or poststates as follows:

a) Let \( S_k \) be a prestate and \( S_1 \) be a prestate or a poststate. Let \( \theta \) be some substitution of parameters of \( S_1 \) for parameters of \( S_k \). If \( (S_1, \theta) \models S_1 \) (improper set inclusion) for some \( \theta \), then merge \( S_k \theta \) into \( S_1 \) (i.e., draw an arc from \( S_k \) to \( S_1 \theta \) for each arc from \( S_k \) to \( S_m \) for all \( m \) and label the arc accordingly; remove \( S_k \) along with its arcs). This step is repeated until no merge is possible.
b) Let $S_k$, $S_1$ be two poststates. If $(S_k \emptyset) = S_1$ merge $S_k \emptyset$ into $S_1$. Repeat this step until no merge is possible.

vii) At this stage, we should have a transition diagram for the insert and delete operations.

\[ \textbf{5.7.4 The Example} \]

As an illustration, we use the static constraints $sc_1$, $sc_2$ defined as follows (see Sect.4.7.2):

\[ sc_1: \ (\forall x)(\forall y)(E(x,y) \implies y > 20,000) \]

where $E(x,y)$ means that $x$ is an employee with salary $y$.

\[ sc_2: \ (\forall x)(\exists y)(M(x) \implies E(x,y)) \]

where $M(x)$ means that $x$ is a manager.

As operation descriptions, we assume the following:

\[ \text{hire}(\bar{x}): \ \text{always}^+ (\exists y)(E(\bar{x},y), S_1 \vDash (I_y)E(\bar{x},y)) \implies \delta(\text{hire}(\bar{x}), S_1) \vDash (I_y)(E(\bar{x},y) \& y > 20000) \]

It states that "if it was always true in the past (the temporal assertion), and it is true in state $S_1$, then in the state resulting from hiring $x$ in state $S_1$ we will know that $x$ is an employee with some salary $y > 20000".

\[ \text{fire}(\bar{x}): \ \text{always}^- (\exists y)(E(\bar{x},y), S_1 \vDash (I_y)E(\bar{x},y)) \implies \delta(\text{fire}(\bar{x}), S_1) \vDash (I_y)E(\bar{x},y) \]

It states that "if $\bar{x}$ is an employee but not a manager in $S_1$, then firing $\bar{x}$ results in that $\bar{x}$ is not an employee." We may similarly phrase the following operation descriptions:

\[ \text{raise}^*(\bar{x}, 10\% \bar{y}): \ \text{always}^+ (\exists y)(E(\bar{x},\bar{y}), S_1 \vDash (I_y)E(\bar{x},\bar{y})) \implies \delta(\text{raise}^*(\bar{x}, 10\% \bar{y}), S_1) \vDash E(\bar{x}, \bar{y} + 10\% \bar{y}) \& \bar{y} + 10\% \bar{y} > 20000 \]

\[ \text{promote}(\bar{x}): \ \text{always}^+ (\exists y)(E(\bar{x},y), S_1 \vDash (I_y)E(\bar{x},y)) \implies \delta(\text{promote}(\bar{x}), S_1) \vDash M(\bar{x}) \]

\[ \text{demote}(\bar{x}): \ \text{always}^+ (\exists y)(E(\bar{x},y), S_1 \vDash (I_y)E(\bar{x},y)) \implies \delta(\text{demote}(\bar{x}), S_1) \vDash \neg M(\bar{x}) \]

\[ \text{engage}(\bar{x}): \ \text{always}^+ (\exists y)(E(\bar{x},y), S_1 \vDash (I_y)E(\bar{x},y)) \implies \delta(\text{engage}(\bar{x}), S_1) \vDash \neg M(\bar{x}) \]

\[ \text{5.7.5 Analysis of an Insert Operation} \]

We show the method by analysing the hire$(\bar{x})$ operation described in the last section. We assume that $SC_1 = SC = \{ sc_1, sc_2 \}$. The clause form of $SC_1$, $SC_1'$, and the precondition and the postcondition of hire$(\bar{x})$ are as follows:
\( C_{C6} = C_1 = \{ \neg E(x,y) \lor y \geq 20000, \neg M(x) \lor E(x,f(x)) \} \)

\( P = \{ \neg E(x,y) \} \)

\( Q = \{ E(x,f(x)), f(x) > 20000 \} \)

i) Applicability analysis:

Let \( c_1 = \neg E(x,y) \lor y > 20000 \) and \( c_2 = \neg M(x) \lor E(x,f(x)) \). Let \( c_3 = P = \neg E(x,y) \)

The unifiability digraph of \( P \cup \{ c_1, c_2 \} \) is

\[
\begin{array}{c}
2.2/1.1+g_1 \quad 2.2/3.1+g_2 \\
\downarrow \quad \downarrow \\
\text{c}_1 \quad \text{c}_2 \quad \text{c}_3
\end{array}
\]

where \( g_1 = \{ \langle f(x), y \rangle \} \) and \( g_2 = \{ \langle x, x \rangle, \langle f(x), y \rangle \} \).

We see that by the purity principle, \( c_1 \) and \( c_2 \) can be removed from the digraph and hence \( P \cup \{ c_1, c_2 \} \) is consistent. This means that the hire(x) operation can be applied in some legal system state, say \( S_1 \).

The prestate of hire(x) can be characterized as

\[ \text{prestate(hire(x))} = \Psi(PUC_1) = \{ \neg E(x,y), \neg M(x) \} \]

The literal \( \neg M(x) \) is derived from \( c_2 \) and \( c_3 \).

ii) Compute \( \Psi(\text{hire(x)}) \), \( \Psi(QU\Psi(\text{hire(x)})) \) and the poststate. We have

\[ \Psi(\text{hire(x)}) \]

\[ = \{ \alpha : \alpha \in \Psi(PUC_1) \land \{ \alpha \} U Q \text{ is consistent} \} \]

\[ = \{ \neg M(x) \} \]

\[ \Psi(QU\Psi(\text{hire(x)})) \]

\[ = \{ E(x,f(x)), f(x) > 20000, \neg M(x) \} \]

Thus, the poststate of the hire(x) operation is

\[ \text{poststate(hire(x))} \]

\[ = \Psi(QU\Psi(\text{hire(x)})) U \Psi(\text{hire(x)}) \]

\[ = \{ E(x,f(x)), f(x) > 20000, \neg M(x) \} \]

It is obvious that \( \text{poststate}(\text{hire(x)}) U C_{C6} \) is consistent.

iii) Check if the operation description is sufficient for characterize the poststate

\[ \Psi(C_1 U Q) \]
\[ \{ E(\bar{x}, f(\bar{x})), f(\bar{x}) > 20000 \} \subseteq \text{poststate}(\text{hire}(\bar{x})). \]

iv) Thus we have the transition depicted below

\[ \text{hire}(\bar{x}) \rightarrow \text{prestate}(\text{hire}(\bar{x})) \rightarrow \text{poststate}(\text{hire}(\bar{x})) \]

v) Analyse the other operation descriptions. The result of the analysis is summarized as follows:

<table>
<thead>
<tr>
<th>prestate</th>
<th>poststate</th>
</tr>
</thead>
<tbody>
<tr>
<td>hire</td>
<td>( S_1 = { \neg E(\bar{x}, \bar{y}), ) \not M(\bar{x}) } )</td>
</tr>
<tr>
<td></td>
<td>( S_2 = { E(\bar{x}, f(\bar{x})), ) \not f(\bar{x}) &gt; 20000, \not M(\bar{x}) } )</td>
</tr>
<tr>
<td>fire</td>
<td>( S_3 = { E(\bar{x}, f(\bar{x})), ) \not f(\bar{x}) &gt; 20000, \not M(\bar{x}) } )</td>
</tr>
<tr>
<td></td>
<td>( S_4 = { E(\bar{x}, \bar{y}), ) \not f(\bar{x}) &gt; 20000, \not ) M(\bar{x}) } )</td>
</tr>
<tr>
<td>promote</td>
<td>( S_5 = { E(\bar{x}, f(\bar{x})), ) \not f(\bar{x}) &gt; 20000, \not M(\bar{x}) } )</td>
</tr>
<tr>
<td></td>
<td>( S_6 = { E(\bar{x}, f(\bar{x})), ) \not f(\bar{x}) &gt; 20000, M(\bar{x}) } )</td>
</tr>
<tr>
<td>demote</td>
<td>( S_7 = { E(\bar{x}, f(\bar{x})), ) \not f(\bar{x}) &gt; 20000, \not M(\bar{x}) } )</td>
</tr>
<tr>
<td></td>
<td>( S_8 = { E(\bar{x}, f(\bar{x})), ) f(\bar{x}) &gt; 20000, \not M(\bar{x}) } )</td>
</tr>
</tbody>
</table>

**Fig. 22. The prestates and the poststates of the operations**

Note that the engage operation cannot pass substep ii) and we choose to remove the engage operation description from our study. We see that \( S_1 \) can be merged into \( S_4 \) (but not \( S_1 \) into \( S_1 \)). \( S_5 \), \( S_9 \) and \( S_2 \) can be merged into \( S_6 \) and \( S_7 \) can be merged into \( S_8 \). We have the following diagram:

\[ \text{demote}(\bar{x}) \]

\[ \text{hire}(\bar{x}) \rightarrow \text{promote}(\bar{x}) \rightarrow \text{fire}(\bar{x}) \]

**Fig. 23. The transition diagram for the insert and delete operations**

### 5.7.5 Formal Analysis of Update Operations

For each update operation \( \langle \text{op} \rangle \) with precondition \( P \) and postcondition \( Q \), perform the following substeps:

1) \( \langle \text{op} \rangle \) is applicable iff there is a state \( S_i \) of the transition diagram and some substitution \( \theta \) such that \( S_i \mid \theta P \). As discussed in Sect. 5.7.1, \( \theta = \{(t_1, x_1), \ldots, (t_n, x_n)\} \) is a substitution of terms
\( t_k \) of \( S_i \) for parameters \( \bar{x}_k \) of \( P \) such that \( k \neq 1 \) implies that \( \bar{x}_k \neq \bar{x}_1 \), for \( k, l = 1, \ldots, n \). \( S_i \) is the prestate of \( \langle \text{op} \rangle \).

ii) The application of \( \langle \text{op} \rangle \) yields some legal system state iff there is a state \( S_j \) of the transition diagram and some extended substitution \( \bar{\theta} \) as described in Sect. 5.7.1 such that \( S_j \bar{\theta} \models O \). \( S_j \bar{\theta} \) is called a poststate of the operation.

iii) Usually there may be more than one \( S_j \) which satisfies the criterion defined in the last step. To identify the desirable \( S_j \), we consider the frame problem again. Recall that an update operation changes some of the terms of the prestate, it should be clear that

\[
\psi(\langle \text{op} \rangle) = \text{prestate}(\langle \text{op} \rangle) - \{ \alpha : \alpha \in \psi(\text{PUC}_1) \bar{\theta} \wedge \alpha \bar{\theta} \in \psi(\text{Q}) \}
\]

denotes those that should remain true in the poststate. Therefore, the frame problem requires that

\[
\psi(\langle \text{op} \rangle) \models \text{poststate}(\langle \text{op} \rangle)
\]

### 5.7.1 Example: The Analysis of an Update Operation

We show the method by analysing the update operation \( \text{raise}^*(\bar{x},10\% \bar{y}) \):

\[
\text{raise}^*(\bar{x},10\% \bar{y}) : \\exists \bar{x}, S_i \models \text{E}(\bar{x}, \bar{y}) \\
\implies \delta(\text{raise}^*(\bar{x},10\% \bar{y}), S_i) \models \text{E}(\bar{x}, \bar{y}+10\% \bar{y}) \wedge \bar{y}+10\% \bar{y} > 20000
\]

i) The operation is applicable in states \( S_8 \) and \( S_6 \), since

\( S_8 \models \text{E}(\bar{x}, \bar{y}) \bar{\theta}_1 \)

where \( \bar{\theta}_1 = \{ \langle f(\bar{x}), \bar{y} \rangle \} \) and

\( S_6 \models \text{E}(\bar{x}, \bar{y}) \bar{\theta}_2 \)

where \( \bar{\theta}_2 = \{ \langle f(\bar{x}), \bar{y} \rangle \} \).

(see Fig. 22).

ii) The application of the operation yields states \( S_8 \) and \( S_6 \) since

\( S_8 \bar{\theta}_1 \models \text{E}(\bar{x}, \bar{y}+10\% \bar{y}) \wedge \bar{y}+10\% \bar{y} > 20000 \)

where \( \bar{\theta}_1 = \{ \langle \bar{y}+10\% \bar{y}, f(\bar{x}) \rangle \} \) and

\( S_6 \bar{\theta}_2 \models \text{E}(\bar{x}, \bar{y}+10\% \bar{y}) \wedge \bar{y}+10\% \bar{y} > 20000 \)

where \( \bar{\theta}_2 = \{ \langle \bar{y}+10\% \bar{y}, f(\bar{x}) \rangle \} \)

iii) \( \psi(\text{raise}^*(\bar{x},10\% \bar{y})) \)

\[
= S_8 - \{ \alpha : \alpha \in \psi(\text{PUC}_1) \bar{\theta}_1 \wedge \alpha \bar{\theta}_1 \in \psi(\text{Q}) \} \\
= S_8 - \{ \text{E}(\bar{x}, f(\bar{x})), f(\bar{x}) > 20000 \}
\]
\[ \{ E(\bar{x}, f(\bar{x})), f(\bar{x}) > 20000, \neg M(\bar{x}) \} = \neg M(\bar{x}) \\]
\[ \{ E(\bar{x}, f(\bar{x})), f(\bar{x}) > 20000 \} \]
\[ \varepsilon(\{ E(\bar{x}, f(\bar{x})), f(\bar{x}) > 20000, \neg M(\bar{x}) \}) \]
\[ S_8 \bar{\theta} \]
\[ \neq S_8 \bar{\theta} \]

where

\[ S_8 \bar{\theta} = \{ E(\bar{x}, f(\bar{x})), f(\bar{x}) > 20000, M(\bar{x}) \} \]

Similarly, the application of \( \text{raise}^* (\bar{x}, 10\bar{y}) \) in state \( S_6 \) yields state \( S_5 \). Thus, the analysis of the \( \text{raise}^* (\bar{x}, 10\bar{y}) \) adds two transitions to the state transition diagram of Fig.23. The resulting transition diagram is depicted below.

**Fig. 24. The transition diagram after analysing the update operation**

In Fig.24, \( \text{raise}^* (\bar{x}, 10\bar{y})/\bar{\theta} \) means that all the states \( S_j \) reachable through the update operation should be denoted as \( S_j \bar{\theta} \). For instance, if we apply the two operations \( \text{raise}^* \), \( \text{promote}^* \) consecutively to \( S_8 \) without considering the update effect, then it will result in states \( S_8, S_6 \). However, because of the update effect, we must consecutively apply the substitutions \( \bar{\theta}, \bar{\theta} \) to these two states. That is, the two states resulting from applying \( \text{raise}^* \), \( \text{promote}^* \) consecutively to \( S_8 \) should be denoted as \( S_8 \bar{\theta} \) and \( S_8 \bar{\theta} \).

**5.8 Summary**

In this section, we have presented an improved method for testing the consistency of static constraints and an improved method for analysing the operation descriptions.

The improved method for testing the consistency of the static constraints is based upon the work by Lewis [59]. In Sect.5.2, we have proposed a definition of compact digraph for a set of clauses. We prove that our definition is equivalent to the compactness of a set of clauses defined by Lewis. Thus, the compactness of a set of clauses can be easily determined from the digraph for the set of clauses.

In Sect.5.3, we have presented some results about the unifiability digraph. These results can be directly used to determine the consistency of a set of clauses in some cases, as we have shown in Sect.5.4 through Sect.5.6. On the other hand, in case the consistency cannot be determined from the unifiability digraph, then a theorem...
prover must be used. Since a compact set of clauses has the property that only finite many new clauses can be produced, the theorem prover will report the inconsistency or the consistency within finite many steps. In Appendix C, we also include a method for testing the consistency of a set of clauses which is based on the work by Prawitz and Robinson [75] [78], respectively.

The analysis of the operation descriptions can also be supported by an existing theorem-prover, e.g., the resolution principle. In Sect.5.7, we have shown that the improved operation analysis method results in the same state transition diagram, which is used to verify the consistency of the temporal constraints. Since the verification of the temporal constraints is the same as that which is used in the modified tableaux approach, we have not discussed this aspect in this chapter.
CHAPTER 6:

CONCLUSION
6 CONCLUSION AND FUTURE RESEARCH DIRECTIONS

6.1 Claimed Contributions

1) In this thesis, we have tried to clarify some of the notions and problems concerning information systems specification and verification. The importance of consistency checking of information systems specification is addressed.

2) Based upon the study of some previous works, we have proposed in this thesis a new classification of the existing models and approaches for information modelling and information systems specification. We believe that the new classification reflects more faithfully and adequately the development of information modelling and systems specification. Indeed, the temptation of modelling reality has been gone through one state (static model) to two states (dynamic models) to more than two states (temporal models) and finally to infinite number of states (full time perspective models). From a chronological point of view, the research in information modelling has been focused on static models by the mid-1970’s; dynamic models were proposed during the late 1970’s and temporal/time perspective models begin in the 1980’s.

3) In this thesis, we have proposed a temporal framework for information system specification and verification. The framework has a sound and complete mathematical basis.

4) Two approaches for verifying an information systems specifications are presented in full detail in Chapt.4 and Chapt.5. To the knowledge of this author, it is the first research work which tries to formally verify the consistency of information systems specifications along the temporal dimension. Previous works such as [62] [64] [5] [58] [15] have not treated the temporal dimension extensively. Further, the consistency checking methods that are proposed in the above mentioned works are meant to detect the inconsistency of the specification only.

The modified tableaux approach has been adapted to the specification and verification of communication protocols by Hove [49]. In this report, the specification of a communication protocol consists of a set of rules, which specifies a legal state of the communication system, a set of operation descriptions which specifies the behavioral aspects of the communication sites. The modified tableaux approach has been implemented in PROLOG by Hove. The implementation has been experimented with the specification and verification of the alternating bit protocol. The result from the experiment is almost the same as the result that is produced manually [49].

5) In mathematical aspect, we have tried to adapt the tableaux approach to consistency checking. A method for formally constructing the system states is proposed.
In Chapt.5, we have define the unifiability digraph which is very useful for consistency checking purpose. We have also proved that our definition is equivalent to the one that is proposed by Lewis [59].

In Appendix C, we propose a method for consistency checking which is based on the works by Prawitz and Robinson respectively [75] [76]. That is, instead of using complex constants and dummies, we use the concept of unification which we believe is an improvement of Prawitz's work.

6.2 Limitations and Future Research Directions

1) In this thesis, we only address the specification problem of information systems, the modelling task is not considered. The assumption of this thesis is that certain modelling approach should be used prior to the specification of the information system.

2) The specification framework is based on mathematical concepts and hence its user interface is poor. As a future research direction, a more user oriented version of the framework is needed.

3) The temporal language that is proposed in this thesis for specifying the temporal constraints and the temporal assertions of the operation descriptions is not powerful enough. Events, tense operators, time points and time intervals have not been considered. In these aspects, the works proposed in [20] [79] [84] can suggest useful extensions to the present framework.

4) Computational complexity of the consistency checking approaches has not been addressed in this thesis. We believe that it should not be difficult to derive the upper bound and average number of unit operations for the modified tableaux approach for a given input. However, it is very difficult to derive such formulae for the improved approach proposed in Chapter 5.

5) In this thesis, we have presented the consistency checking framework in a straightforward manner. Various heuristic information and more intelligent strategies should be used in practice.
REFERENCES


[56] Langefors, B., Theoretical aspects of information systems for management, IFIP Congress 1974. pp.337--345. (The same paper as [55]).


APPENDIX A: TRANSFORMATION TO CLAUSE FORM

In order to present the algorithm for testing the consistency of a set of wffs, we need the concept of clause form. This concept is used extensively in theorem proving, see e.g. [75] [78] [36].

A formula is said to be in clause form if it is of the form

\[ \alpha_1 \lor \alpha_2 \lor \ldots \lor \alpha_n \]

where \( \alpha_i \), \( i=1, \ldots, n \) is called a literal and it is either an atomic formula or the negation of an atomic formula. In the resolution method, it is required that all the formulae be in clause form. We show in what follows how a formula can be put into clause form.

As an example, we use the following formula to illustrate the process. (For clarity, we use also square brackets in addition to parentheses.)

\[
(\exists y)(E(y) \land \neg (\exists y)M(y)) \land (\forall x)(E(x) \rightarrow (\exists y)WF(x,y))
\]

& \( (\forall x)([\exists y)WF(y,x) \rightarrow M(x)]) \)

An intuitive interpretation of the formula could be that:

a) there is someone who is an employee and there exists no one who is a manager;

b) every employee works for some one; and

c) any body whom is worked for by some body must be a manager.

The process of putting a formula into clause form consists of 7 steps [72]:

1) Eliminate implication signs: Since the clause form of a formula has no occurrence of the \( \rightarrow \) (implication) connective, we replace all the occurrences of \( w_1 \rightarrow w_2 \) by \( \neg w_1 \lor w_2 \) throughout the formula. Apply this to our example formula yields

\[
(\exists y)(E(y) \land \neg (\exists y)M(y)) \land (\forall x)(\neg E(x) \lor (\exists y)WF(x,y))
\]

& \( (\forall x)(\neg (\exists y)WF(y,x) \lor M(x)]) \)

2) Reduce the scope of negation signs. The clause form of a formula requires that the negation sign is applied to only atomic formulae. Therefore we repeatedly apply the following equivalences until each \( \neg \) is applied to an atomic formula:

\[ \neg (w_1 \land w_2) \iff \neg w_1 \lor \neg w_2 \]

\[ \neg (w_1 \lor w_2) \iff \neg w_1 \land \neg w_2 \]

\[ \neg \neg w_1 \iff w_1 \]
- $\neg(\forall x)w_1 \iff (\exists x)\neg w_1$

- $\neg(\exists x)w_1 \iff (\forall x)\neg w_1$

The example after this step becomes

$$(\exists y)[E(y) \& (\forall y)\neg M(y)] \land (\forall x)[\neg E(x) \lor (\exists y)WF(x, y)]$$

$$\land (\forall x)[(\forall y)\neg WF(y, x) \lor M(x)]$$

3) Standardize variables: rename the variables so that each variable in the formula is quantified by a single quantifier. In our example we see that the variable $y$ in $M(y)$ is quantified by two quantifiers. We may rename it as $z$ so that every variable is now quantified by only one quantifier.

$$(\exists u)[E(u) \& (\forall z)\neg M(z)] \land (\forall x)[\neg E(x) \lor (\exists y)WF(x, y)]$$

$$\land (\forall v)[(\forall w)\neg WF(w, v) \lor M(v)]$$

4) Remove existential quantifiers: Consider the wff

$$(\forall x)(\exists y)HAS\text{-}FATHER(x, y)$$

which expresses that for every (individual) $x$, there exists an (individual) $y$ such that $x$ has as father $y$. Clear, not every individual can be the father of a given individual. Moreover, the father of a given individual is unique although several individuals may have the same individual as the father. This situation resembles a functional dependence in mathematics. That is, we define $father(x)$ as the individual who is the father of $x$. Such a function is commonly called a Skolem function. If we use the Skolem function in place of the $y$ that exists, we can eliminate the existential quantifier and write

$$(\forall x)HAS\text{-}FATHER(x, father(x))$$

Clearly, if the existential quantifier is within the scopes of $n$ universal quantifiers, then the Skolem function will have $n$ arguments.

By this method, we can remove the existential quantifiers from our example. Note that if an existential quantifier is not within the scope of any universal quantifier, then we just simply introduce a zero-ary function (i.e., a constant symbol) in place of the existentially quantified variable. Clearly the Skolem function symbols that are introduced must be unique. In our example, we eliminate the two existential quantifiers $(\exists y)$ and $(\exists u)$ by introducing $g(x)$ and $a$ as the Skolem functions for the variables $y$ and $u$ respectively.

$$[E(a) \& (\forall z)\neg M(z)] \land (\forall x)[\neg E(x) \lor WF(x, g(x))]$$

$$\land (\forall v)[(\forall w)\neg WF(w, v) \lor M(v)]$$
5) Since all the variables at this stage are universally quantified, we may eliminate all the universal quantifiers from the formula and keep in mind that all the variables are universally quantified. By doing this our example becomes

\[ E(a) \land \lnot M(z) \land (\lnot E(x) \lor WF(x, g(x))) \land (\lnot WF(w, v) \lor M(v)) \]

6) Now we still have to remove the \& signs. This is done in two steps. In this step, we convert the formula into conjunctive normal form and in the next step we eliminate the \& signs.

To convert the formula into conjunctive normal form, we repeatedly apply to the formula the distribution law

\[ w_1 \lor (w_2 \land w_3) \iff (w_1 \lor w_2) \land (w_1 \lor w_3) \]

Our example has already in conjunctive normal form, we have nothing to do in this step.

7) Eliminate the \& signs: removing the \& signs from our example gives us a set of formulae in clause form, that is, it gives rise to a set of clauses:

- \( E(a) \)
- \( \lnot M(z) \)
- \( \lnot E(x) \lor WF(x, g(x)) \)
- \( \lnot WF(w, v) \lor M(v) \)

It can be easily seen from the above that any WFF can be converted to a set of clauses. Given a set of WFFs, we can convert each of the WFF into a set of clauses. This implies that we can convert the set of WFFs into a set of clauses. The conversion as described above has the property that if the original set of WFFs is inconsistent, then the set of clauses that is converted is also inconsistent [78] [72].
APPENDIX B: UNIFICATION AND UNIFICATION PROCESS

As defined in Sect.3.1.1, a term is either a variable, a constant symbol or an m-ary function symbol whose m variables are substituted by m terms. We also defined in Appendix A that a literal is an atomic formula or the negation of an atomic formula which may or may not contain variables. If a literal contains some variables then we can substitute terms for the variables to obtain a "substitution instance" of the literal. In general, we can represent any substitution by a set of ordered pairs

\[ \emptyset = \{ (t_1, x_1), \ldots, (t_n, x_n) \} \]

such that \( x_i = x_j \) then \( i = j \). The pair \( (t_i, x_i) \) means that term \( t_i \) is substituted for variable \( x_i \) throughout the literal.

Let \( \alpha \) be a literal and \( \emptyset \) a substitution, the substitution instance obtained by substitute the terms for the variables using \( \emptyset \) throughout \( \alpha \) is denoted as \( \alpha : \emptyset \).

For instance, if \( \emptyset_1 = \{ (a, x) \} \), then \( \lnot E(x) : \emptyset_1 \) is \( \lnot E(a) \) and \( WF(x, g(x)) : \emptyset_1 \) is \( WF(a, g(a)) \). If \( \emptyset_2 = \{ (g(a), v), (a, w) \} \), then \( \lnot WF(w, v) : \emptyset_2 \) is \( \lnot WF(a, g(a)) \) and \( \lnot E(x) : \emptyset_2 \) is \( \lnot E(x) \).

Similarly, we can denote the result of applying a substitution \( \emptyset \) to a set of literals \( \{ \alpha_i \} \) as \( \{ \alpha_i \} : \emptyset \). A set \( \{ \alpha_i \} \) of literals is unifiable if there exists a substitution \( \emptyset \) such that \( \alpha_1 : \emptyset = \alpha_2 : \emptyset = \alpha_3 : \emptyset \), etc. In such a case, \( \emptyset \) is said to be a unifier of \( \{ \alpha_i \} \) since its use collapses the set to a singleton. For instance, \( \emptyset = \{ (x, w), (g(x), v) \} \) is a unifier for \( \{ \alpha_1, \alpha_2 \} = \{ WF(x, g(x)), WF(w, v) \} \) because \( \{ \alpha_1, \alpha_2 \} : \emptyset = \{ WF(x, g(x)) \} \). Another unifier for this same set of literals is \( \emptyset_1 = \{ (a, w), (w, x), (g(x), v) \} \) because \( \{ \alpha_1, \alpha_2 \} : \emptyset_1 = \{ WF(a, g(a)) \} \). We see that the unifier for a set of literals is not unique.

Observing the two unifiers in the last paragraph, we see that \( \{ \alpha_1, \alpha_2 \} : \emptyset_1 \) can be obtained from \( \{ \alpha_1, \alpha_2 \} : \emptyset \) by using another substitution \( \emptyset_2 = \{ (a, x) \} \). Since \( \{ \alpha_1, \alpha_2 \} : \emptyset \) is also a set of literals, we may apply \( \emptyset_2 \) to \( \{ \alpha_1, \alpha_2 \} : \emptyset \) in the same way. That is, we may denote the applications of the two substitutions \( \emptyset \) and \( \emptyset_2 \) to \( \{ \alpha_1, \alpha_2 \} \) as \( \{ \alpha_1, \alpha_2 \} : \emptyset : \emptyset_2 \). We see that

\[ \{ \alpha_1, \alpha_2 \} : \emptyset : \emptyset_2 = \{ WF(x, g(x)), WF(w, v) \} : \emptyset : \emptyset_2 \]
\[ = \{ WF(x, g(x)) \} : \emptyset_2 \]
\[ = \{ WF(a, g(a)) \} \]
\[ = \{ \alpha_1, \alpha_2 \} : \emptyset_1 \].

The example shows that \( \emptyset \) has the property that the substitution instance \( \{ \alpha_1, \alpha_2 \} : \emptyset_1 \) can be obtained from \( \{ \alpha_1, \alpha_2 \} : \emptyset \) by another substitution \( \emptyset_2 \).

The above discussion leads to the notion of the "most general unifier" (mgu). A substitution \( \emptyset \) is said to be the mgu of a set of literals \( \{ \alpha_i \} \) if for any unifier \( \emptyset_1 \) of \( \{ \alpha_i \} \) which yields \( \{ \alpha_i \} : \emptyset_1 \), there exists a substitution \( \emptyset_2 \) such that \( \{ \alpha_i \} : \emptyset_1 = \{ \alpha_i \} : \emptyset_2 \). Furthermore, the common
substitution instance that is produced by a mgu is unique except for renaming the variables.

There is an algorithm called the unification algorithm that produces a mgu \( \theta \) for any set \( \{ \alpha_i \} \) of unifiable literals and reports failure if the set is not unifiable. The algorithm can be outlined as follows:

1) Let \( \theta_0 \) be the empty substitution.

2) Suppose that \( \theta_j \) has been constructed. If all the literals in \( \{ \alpha_i \}:\theta_j \) become identical, then let \( \theta = \theta_j \) and \( \theta \) is the mgu desired. Otherwise go to 3).

3) Since not all of the literals in \( \{ \alpha_i \}:\theta_j \) are identical, the algorithm tries to detect the first symbol position in which not all of the literals in \( \{ \alpha_i \}:\theta_j \) have the same symbol. It then constructs a disagreement set containing the "well-formed" expressions from each literal that begins with this position. A well-formed expression is either a term or a literal. That is, a disagreement set contains a well-formed expression from each of the literals of \( \{ \alpha_i \}:\theta_j \) such that the first symbol of some of these well-formed expressions does not agree with each other. In order to collapse \( \{ \alpha_i \}:\theta_j \) to a singleton, such a disagreement must be removed. This can be done only if some sequence of substitutions \( \theta_{j+1}, \theta_{j+2}, \ldots, \theta_{j+n} \), for some \( n \) can be defined so that \( \{ \alpha_i \}:\theta_j, \theta_{j+1}, \ldots, \theta_{j+n} \) becomes a singleton. Clearly,

4) If the disagreement set contains no variables at all, then the algorithm fails to find a mgu, because \( \theta_{j+1} \) cannot be defined. That is, the algorithm stops with failure. Otherwise,

5) Let \( x_j \) be any variable and \( t_j \) be any term (possibly another variable) in the disagreement set such that \( t_j \) does not contain \( x_j \). (If no such \( t_k \) exists, then the algorithm also fails.) Let \( \theta_{j+1} = \theta_j \cup \{ (t_j, x_j) \} \) and go to 2).

It has been proved by Robinson [78] and Luckham [61] that the unification algorithm as described above finds a mgu of a set of unifiable literals and reports failure when the literals are not unifiable.

As an example, we show the construction of the mgu of the set of literals

\[
\{ \text{WF}(x,g(x)), \text{WF}(w,v) \}\]
<table>
<thead>
<tr>
<th>step</th>
<th>substitution</th>
<th>${\alpha_i}:\emptyset$</th>
<th>disagreement set</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[]</td>
<td>{WF(w,v), WF(x,g(x))}</td>
<td>{w, x}</td>
</tr>
<tr>
<td>1</td>
<td>{(x,w)}</td>
<td>{WF(x,v), WF(x,g(x))}</td>
<td>{v, g(x)}</td>
</tr>
<tr>
<td>2</td>
<td>{(x,w),(g(x),v)}</td>
<td>{WF(x,g(x))}</td>
<td>{}</td>
</tr>
</tbody>
</table>

Thus, the mgu is \{(x,w),(g(x),v)\}.
APPENDIX C: AN IMPROVED PROOF METHOD

1. Prawitz's Work

In [75], a method for proving the validity of a first order wff is presented. The method can be outlined as follows, with modifications according to our convenience.

Given a sequent

(1) $w_1, \ldots, w_i, w_{i+1}, \ldots, w_n$

where $w_1, \ldots, w_n$ are closed wffs. It is required to prove the validity (i.e., satisfied in all models) of

(2) $w_1 \land \ldots \land w_i \rightarrow w_{i+1} \lor \ldots \lor w_n$

That is, to prove that if all the $w_1, \ldots, w_i$ are true then at least one of the $w_{i+1}, \ldots, w_n$ is true.

It is equivalent to prove that

(3) $\neg w_1 \lor \ldots \lor \neg w_i \lor w_{i+1} \lor \ldots \lor w_n$

In order to do this, we may first translate each of $\neg w_1, \ldots, \neg w_i, w_{i+1}, \ldots, w_n$ into its prenex normal form. The resulting formula may be denoted as

(4) $w_1', \ldots, w_i', w_{i+1}', \ldots, w_n'$

where the commas is to be understood as logical or.

A universal quantifier in (4) is called a constant generating quantifier (C-quantifier) and, an existential quantifier in (4) is called a dummy generating quantifier (D-quantifier).

Step 1. Make the following transformation of every formula in (4):
Delete every C-quantifier (with its attached variable) not preceded by a D-quantifier and substitute a constant symbol for all (remaining) occurrences of the quantified variable. We call all the constant symbols that have appeared so far simple constants.

Step 2. Delete every C-quantifier (with its attached variable) and substitute a constant $c[p,q,1]$ for all remaining occurrences of the variable. The constant $c[p,q,1]$ is called a complex constant and is to be such that: $p$ denotes the number of the formula in which the quantifier occurs. $q$ denotes the number of the quantifier in the formula, counting the quantifiers from the left to the right. $1$ indicates that the constant is introduced in cycle 1 of the computation.

In the same way, we delete every D-quantifier (with its attached variable) and substitute a dummy $d[p,c,1]$ for all (remaining) occurrences of the quantified variable.

We now have $n$ quantifier-free formulae.
(5) \( w_1', \ldots, w_n' \)

(5) is called the origin formula which will be abbreviated as \( \text{OF} \).

Step 3. This step is performed iteratively. Each iteration is called a cycle. When starting a new cycle we call it the \( s \)-th cycle if the previous cycle was the \((s-1)\)-th cycle. Let the cycle to be described be the \( t \)-th cycle \((t = 1, 2, \ldots)\).

Phase 1. Transform an instance of the origin formula, \( \text{OF} \), by substituting \( t \) for the third index in all the dummies an complex constants. The resulting formula is called \( \text{OF}_t \).

Phase 2. Form the \( t \)-th cycle formula, abbreviated: \( \text{CF}_t \), defined as follows. The first cycle formula is \( \text{OF}_1 \) transformed to conjunctive normal form. If the \( s \)-th cycle formula is \( \text{CF}_s \), then the \((s+1)\)-th cycle formula is \(( \text{CF}_s \lor \text{OF}_{s+1} )\) transformed to conjunctive normal form.

Phase 3.1. If there is a conjunction clause of \( \text{CF}_t \) in which no predicate symbol (with any number of arguments) occurs both negated and not negated, then the given sequent (2) is not provable and the procedure is concluded. Otherwise we continue.

Phase 3.2. We now associate an identity condition, \( \text{IC}_i \), with every conjunction clause, \( M_i \) of \( \text{CF}_t \) \((i = 1, 2, \ldots, m)\), where \( m \) is the number of conjunction clauses of \( \text{CF}_t \). \( \text{IC}_i \) is formed as follows.

1) If \( M_i \) contains two formulae one of which is the negation of the other, then \( \text{IC}_i \) is \( c_1 = c_1' \).

2) If \( M_i \) does not contain two formulae as in 1), then for every couple of atomic formulae \( P(t_1, \ldots, t_k) \) and \( \neg P(t_1', \ldots, t_k') \) in \( M_i \), where \( P \) is a predicate symbol of \( k \)-place, and \( t_1, \ldots, t_k, t_1', \ldots, t_k' \) are constants or dummies. We form an identity list, \( \text{IL}_{ij} \) \((j = 1, 2, \ldots, n_i; n_i \) is assumed to be the number of such couples occurring in \( M_i \)) of the form:

\[
(6) \quad \{ t_1 = t_1' \land t_2 = t_2' \land \ldots \land t_k = t_k' \}
\]

\( \text{IC}_i \) is to be the disjunction of all the \( n_i \) identity lists:

\[
(7) \quad \text{IL}_{11} \lor \text{IL}_{12} \lor \ldots \lor \text{IL}_{1n_i}
\]

Phase 4. The sequent (2) is provable in cycle \( t \) if and only if the conjunction of the identity conditions

\[
(8) \quad \text{IC}_1 \land \text{IC}_2 \land \ldots \land \text{IC}_m
\]

\[
= (\text{IL}_{11} \lor \text{IL}_{12} \lor \ldots \lor \text{IL}_{1n_1})
\]

\[
\land (\text{IL}_{21} \lor \text{IL}_{22} \lor \ldots \lor \text{IL}_{2n_2})
\]

\[
\land \ldots
\]

\[
\land (\text{IL}_{m1} \lor \text{IL}_{m2} \lor \ldots \lor \text{IL}_{mnm})
\]

does not contradict the following three restrictions:
1) Restriction 1. Two different constants are never equal if they are not complex and differ from each other only in respect to the third index. That is, $c_k = c_{k'}$ then $k = k'$. $c_k \neq c[p,q,s]$. $c[p,q,s] \neq c[p',q',s']$ if not $p=p'$ and $q=q'$.

2) Restriction 2. If $c[p,q,s] = c[p,q,s']$, then for all $x < q$, $d[p,x,s] = d[p,x,s']$

3) Restriction 3. If $q < q'$, then $d[p,q,s] \neq c[p,q',s]$.

If (8) contradicts one of these restrictions, then we start another new cycle until we can prove (2).

The idea behind the method is rather clear. Steps 1 and 2 are just transformations using logic equivalences.

In Step 3, we first construct the cycle formula $CF_t$ which is a conjunction of conjunction clauses. Each conjunction clause is a disjunction of atomic formulae or negated atomic formulae. Clearly, if every conjunction clause is true under the assignment of (8) and (8) does not contradict the three restrictions, then we must have a system of formulae for the dummies which makes $CF_t$ true. This is in turn means that (2) must be true.

In Phase 3.1, we examine if there is a conjunction clause of $CF_t$ in which no predicate symbol occurs both negated and not negated. If there is such a clause, then clearly this conjunction clause can never be provable and hence $CF_t$ can never be provable.

The three restrictions are also easy to understand. Restriction 1 says that two (simple) constants can never be equal, which is obvious. Further, two complex constants can be equal if they differ only in the third index. If we regard complex constants as Skolem functions, then this restriction means that the values of two Skolem functions can never equal if the Skolem function symbols are different or their arguments are different. It is also obvious from the definition of Skolem functions.

In a similar way, Restrictions 2 complies the rule for defining the Skolem functions. That is, if the values of two instances of a Skolem function are equal, then all the arguments of the Skolem function must be universally quantified and preceed the existential quantifier for which the Skolem function is introduced.

Restriction 3 complies to the rule of substituting terms to variables. That is, a term is not allowed to substitute for a variable if the term contains the variable.

Indeed, the above observation is important for developing our method in the next section, where we will use Skolem functions instead of complex constants in the procedure.

2. Our Method

Based upon the work of Prawitz, we formulate an improved method for testing the inconsistency of a set of wffs in clause form.
To facilitate understanding, we will use the following example:

C1: \( \neg WF(u, \text{Chief-manager}) \vee WF(Tom, u) \)

Tom works for anybody who works for the Chief-manager.

C2: \( \neg WF(x, x) \vee \neg WF(y, x) \)

Nobody works for some body who works for himself.

C3: \( WF(Tom, \text{Chief-manager}) \)

Tom works for the Chief-manager.

Step 1. Let \( S = \{ C_1, C_2, \ldots, C_n \} \) be a set of clauses, where each \( C_i, i = 1, \ldots, n \), is of the form \( \alpha_{i_1} \vee \alpha_{i_2} \vee \ldots \vee \alpha_{i_m} \) where \( \alpha_{i_j}, j = 1, \ldots, m_i \), is a literal. If \( m_i = 0 \), then \( S \) contains an empty clause and hence \( S \) is inconsistent. Otherwise, define the disjunction normal form of \( C_1 \) & \( \ldots \& C_n \) as the origin formula OF. If OF contains a disjunction clause which contains an atomic formula as well as the negation of the same atomic formula, then the entire disjunction clause can be removed from OF without affecting the consistency of \( S \). Because of this, we will assume in the sequel that OF contains no such disjunction clause.

The OF for our example is

\[
\begin{align*}
&\neg WF(u, \text{Chief-manager}) \& \neg WF(x, x) \& WF(Tom, \text{Chief-manager}) \\
\vee &\neg WF(u, \text{Chief-manager}) \& \neg WF(y, x) \& WF(Tom, \text{Chief-manager}) \\
\vee &WF(Tom, u) \& \neg WF(x, x) \& WF(Tom, \text{Chief-manager}) \\
\vee &WF(Tom, u) \& \neg WF(y, x) \& WF(Tom, \text{Chief-manager})
\end{align*}
\]

Step 2. In cycle \( t \) we mark every variable of the origin formula OF by superscribed \( t \). The resulting formula is denoted as \( OF_t \). That is, the \( OF_t \) of our example is

\[
\begin{align*}
&\neg WF(u^t, \text{Chief-manager}) \& \neg WF(x^t, x^t) \& WF(Tom, \text{Chief-manager}) \\
\vee &\neg WF(u^t, \text{Chief-manager}) \& \neg WF(y^t, x^t) \& WF(Tom, \text{Chief-manager}) \\
\vee &WF(Tom, u^t) \& \neg WF(x^t, x^t) \& WF(Tom, \text{Chief-manager}) \\
\vee &WF(Tom, u^t) \& \neg WF(y^t, x^t) \& WF(Tom, \text{Chief-manager})
\end{align*}
\]

Step 3. Form the \( t \)-th cycle formula \( CF_t \) as follows. \( CF_t \) is \( OF_t \) which has been defined above. If the \( s \)-th cycle formula is \( CF_s \) then the \( (s+1) \)-th cycle formula is \( \{ CF_s \& OF_{s+1} \} \) transformed into disjunctive normal form. This step corresponds to Phase 2 of Step 3 of Prawitz.

Step 4. If there is a disjunction clause of \( CF_t \) in which no literal is unifiable with the negation of any other literal, then \( S \) is consistent. Suppose that there is such a disjunctive clause of \( CF_t \), let it be

\[
\alpha_{i_{k_1}} \& \ldots \& \alpha_{i_{n_k}}.
\]

Then there must be the disjunction clause

\[
\alpha_{i_{k_1}} \& \ldots \& \alpha_{i_{n_k}}.
\]

in \( CF_t \). A model of \( S \) will be

\[
\{ \alpha_{i_{k_1}} \theta : \alpha_{i_{k_1}} \theta \in \mathcal{UH}(S) \}
\]

where \( \mathcal{UH}(S) \) is the Herbrand expansion of \( S \) and \( \mathcal{UH}(S) \) is the set of all literals appearing in clauses of \( \mathcal{H}(S) \). This step corresponds to Phase 3.1 of Prawitz's procedure. However, the condition that is defined here is much weaker than the one that is defined in Prawitz's work
(Conf. Phase 3.1 in the last section).

If there is no such a disjunction clause as described above, we continue. We associate an identity condition $IC_i$ with every disjunctive clause $M_i$ of $CL_i$ as follows.

1) If $M_i$ contains two formulae one of which is the negation of the other, then $IC_i$ is $c_i = c_{i1}$

2) If $M_i$ is not as in 1), then for every couple of literals in $M_i$, we examine if one of them is unifiable with the negation of the other. (This can be easily achieved by the unifiability digraph developed earlier.)

If $\alpha_i$ and $\beta_i$ of $M_i$ is unifiable, and $\theta$ is the most general unifier for $\alpha_i$ and $\beta_i$, then the identity list is simply $\theta$.

$IC_i$ is to be the disjunction of all the $n_i$ identity lists. $n_i$ is supposed to be the number of couples of literals in $M_i$ such that one of which can be unifiable with the negation of the other. This corresponds to Phase 3.2 of Prawitz.

As an illustration, our example in cycle 1 has the following four disjunction clauses $M_1$, $M_2$, $M_3$, and $M_4$ as shown below.

$M_1$: \( \lnot WF(u_1, \text{Chief-manager}) \land WF(x_1, x_1) \land WF(\text{Tom}, \text{Chief-manager}) \)

$M_2$: \( \lnot WF(u_1, \text{Chief-manager}) \land WF(y_1, x_1) \land WF(\text{Tom}, \text{Chief-manager}) \)

$M_3$: \( WF(\text{Tom}, u_1) \land \lnot WF(x_1, x_1) \land WF(\text{Tom}, \text{Chief-manager}) \)

$M_4$: \( WF(\text{Tom}, u_1) \land \lnot WF(y_1, x_1) \land WF(\text{Tom}, \text{Chief-manager}) \)

The identity conditions for the $M_i$'s, $i = 1, \ldots, 4$, are as follows. Note that we have rewritten the mgu in the form of identity. For example, the mgu for $\lnot WF(u_1, \text{Chief-manager}) \land WF(\text{Tom}, \text{Chief-manager})$ of $M_1$ is \{ (Tom, $u_1$) \}, which is rewritten as $u_1 := \text{Tom}$ in the following $IC_i$'s. (Note that $::=$ is not symmetric, that is, $x := y$ does not necessarily imply $y := x$. However, $::=$ is transitive, i.e., if $x ::= y$ and $y ::= z$, then $x ::= z$)

$IC_1$: $u_1 := \text{Tom}$

$IC_2$: \( (u_1 := \text{Tom}) \lor (x_1 := \text{Chief-manager} \land y_1 := \text{Tom}) \)

$IC_3$: $x_1 := \text{Tom} \land u_1 := x_1$

$IC_4$: \( (y_1 := \text{Tom} \land u_1 := x_1) \lor (x_1 := \text{Chief-manager} \land y_1 := \text{Tom}) \)

Intuitively, $IC_1$ indicates that under the substitution of Tom to $u_1$ $M_1$ will be false. $IC_2$ indicates that either substituting Tom for $u_1$ or substituting Chief-manager for $x_1$ and Tom for $y_1$ will make $M_2$ become false. We can similarly interpret $IC_3$ and $IC_4$. Clearly, if there is a combination of one identity list from each of the $IC_i$'s such that there exists no contradictory assignment of values to the variables, then each of the $M_i$'s will be false. That is, the disjunction of the $M_i$'s will also be false which in turn indicates that the set $S$ of clauses is inconsistent.

Step 5. $S$ can be confuted in cycle $s$ iff the conjunction of the identity conditions
IC₁ & IC₂ & ... & ICₙ
= (IL₁₁ V ... V IL₁₁₁) & ... & (ILₘ₁ V ... V ILₘₙₙ)
contains a disjunction clause
(9) IL₁q₁ & IL₂q₂ & ... & ILₘqₘ
(where 1 ≤ q₁ ≤ n₁, 1 ≤ q₂ ≤ n₂, ..., 1 ≤ qₘ ≤ nₘ)
which does not violate the following restriction:

RESTRICTION. If \( x_{i} := t_{1} \) and \( x_{j} := t_{2} \) can be derived from (9), where \( i \leq s \) and \( t_{1} \) and \( t_{2} \) are terms, then \( t_{1} \) and \( t_{2} \) must agree. \( t_{1} \) and \( t_{2} \) do not agree if either of the following holds; otherwise, \( t_{1} \) and \( t_{2} \) are said to agree:

1) \( t_{1} \) and \( t_{2} \) are two different constant symbols. This is implied by Restriction 1 of Prawitz.

2) \( t_{1} = y \) and \( t_{2} = f(..., y, ...) \), where \( y \) is a variable introduced in some cycle and \( f \) is a function symbol. This is equivalent to Restriction 3 of Prawitz.

3) \( t_{1} \) is a constant symbol and \( t_{2} = f(...) \), where \( f \) is a function symbol. This is implied by Restriction 1 of Prawitz.

4) \( t_{1} = f(...) \) and \( t_{2} = g(...) \) and \( f \neq g \). This is implied by Restriction 1 of Prawitz.

5) \( t_{1} = f(t_{11}, ..., t_{1k}) \) and \( t_{2} = f(t_{21}, ..., t_{2k}) \) and some pair \( \langle t_{1j}, t_{2j} \rangle, j = 1, ..., k \), do not agree. This is implied by Restrictions 1 and 2 of Prawitz.

It can be proved that the above restriction is essentially the same restrictions defined by Prawitz.

If \( S \) cannot be confuted in cycle \( s \), then we start another cycle.

As an illustration, we see that

\[ u_{1} := \text{Tom from } IC_{1} \text{ and} \]

\[ u_{1} := \text{Tom from } IC_{2} \text{ and} \]

\[ u_{1} := x_{1} \text{ and } x_{1} := \text{Tom from } IC_{3} \text{ and} \]

\[ u_{1} := x_{1} \text{ and } y_{1} := \text{Tom from } IC_{4} \]

implies

\[ u_{1} := x_{1} := y_{1} := \text{Tom} \]

which does not violate the restriction defined above. This means that \( \{ C_{1}, C_{2}, C_{3}, C_{4} \} \) is inconsistent.