Towards Effective and Efficient Temporal Verification in Grid Workflow Systems

by

Jinjun Chen

B.Sci. (XiDian University)
M.Eng. (XiDian University)

A thesis submitted to
CITR – Centre for Information Technology Research
Faculty of Information and Communication Technologies
Swinburne University of Technology

for the degree of
Doctor of Philosophy

February 2007
To my parents and brothers
Declaration

This thesis contains no material which has been accepted for the award of any other degree or diploma, except where due reference is made in the text of the thesis. To the best of my knowledge, this thesis contains no material previously published or written by another person except where due reference is made in the text of the thesis.

Jinjun Chen

February 2007
Acknowledgements

I sincerely express my deepest gratitude to my supervisor, Professor Yun Yang, for his seasoned supervision and continuous encouragement throughout the course of this work, and for his careful reading and appraisal of drafts of this thesis. Without his consistent support, I would not have been able to complete this manuscript.

I thank Swinburne University of Technology and the Faculty of Information and Communication Technologies for offering me a full Research Scholarship throughout my doctoral program. I also thank the Research Committee of the Faculty of Information and Communication Technologies for research publication funding support and for providing me with financial support to attend conferences.

My thanks also go to staff members, research students and research assistants at previous CICEC and/or current CITR for their help, suggestions, friendship and encouragement, in particular, Jun Han, Ryszard Kowalczyk, Chengfei Liu, T.Y Chen, Neroli Finlay, Charlotte Swain, Gillian Foster, Gwyn Morgan, Phil Joyce, Peter Eden, Phil Freeman, Baolin Wu, Minh Hong Tran, Joel Lignier and Richard Moore.

Last but not least, I am deeply grateful to my parents and brothers for their love, understanding, encouragement, sacrifice and help.
Abstract

In grid architecture, a grid workflow system is a type of high-level grid middleware which aims to support large-scale sophisticated scientific or business processes in a variety of complex e-science or e-business applications such as climate modelling, disaster recovery, medical surgery, high energy physics, international stock market modelling and so on. Such sophisticated processes often contain hundreds of thousands of computation or data intensive activities and take a long time to complete. In reality, they are normally time constrained. Correspondingly, temporal constraints are enforced when they are modelled or redesigned as grid workflow specifications at build-time. The main types of temporal constraints include upper bound, lower bound and fixed-time. Then, temporal verification would be conducted so that we can identify any temporal violations and handle them in time.

Conventional temporal verification research and practice have presented some basic concepts and approaches. However, they have not paid sufficient attention to overall temporal verification effectiveness and efficiency. In the context of grid economy, any resources for executing grid workflows must be paid. Therefore, more resources should be mainly used for execution of grid workflow itself rather than for temporal verification. Poor temporal verification effectiveness or efficiency would cause more resources diverted to temporal verification. Hence, temporal verification effectiveness and efficiency become a prominent issue and deserve an in-depth investigation.

This thesis systematically investigates the limitations of conventional temporal verification in terms of temporal verification effectiveness and efficiency. The detailed analysis of temporal verification effectiveness and efficiency is conducted
for each step of a temporal verification cycle. There are four steps in total: Step 1 – defining temporal consistency; Step 2 – assigning temporal constraints; Step 3 – selecting appropriate checkpoints; and Step 4 – verifying temporal constraints. Based on the investigation and analysis, we propose some new concepts and develop a set of innovative methods and algorithms towards more effective and efficient temporal verification. Comparisons, quantitative evaluations and/or mathematical proofs are also presented at each step of the temporal verification cycle. These demonstrate that our new concepts, innovative methods and algorithms can significantly improve overall temporal verification effectiveness and efficiency.

Specifically, in Step 1, we analyse the limitations of two temporal consistency states which are defined by conventional verification work. After, we propose four new states towards better temporal verification effectiveness. In Step 2, we analyse the necessity of a number of temporal constraints in terms of temporal verification effectiveness. Then we design a novel algorithm for assigning a series of fine-grained temporal constraints within a few user-set coarse-grained ones. In Step 3, we discuss the problem of existing representative checkpoint selection strategies in terms of temporal verification effectiveness and efficiency. The problem is that they often ignore some necessary checkpoints and/or select some unnecessary ones. To solve this problem, we develop an innovative strategy and corresponding algorithms which only select sufficient and necessary checkpoints. In Step 4, we investigate a phenomenon which is ignored by existing temporal verification work, i.e. temporal dependency. Temporal dependency means temporal constraints are often dependent on each other in terms of their verification. We analyse its impact on overall temporal verification effectiveness and efficiency. Based on this, we develop some novel temporal verification algorithms which can significantly improve overall temporal verification effectiveness and efficiency. Finally, we present an extension to our research about handling temporal verification results since these verification results are based on our four new temporal consistency states.

The major contributions of this research are that we have provided a set of new concepts, innovative methods and algorithms for temporal verification in grid workflow systems. With these, we can significantly improve overall temporal
verification effectiveness and efficiency. This would eventually improve the overall performance and usability of grid workflow systems because temporal verification can be viewed as a service or function of grid workflow systems. Consequently, by deploying the new concepts, innovative methods and algorithms, grid workflow systems would be able to better support large-scale sophisticated scientific and business processes in complex e-science and e-business applications in the context of grid economy.
The Author’s Publications

Books


Journals


**Conferences**


Table of Contents

Chapter 1  Introduction ........................................................................................................1
  1.1 Introduction to temporal verification in grid workflow systems .................1
  1.2 Key issues of this research ...........................................................................5
  1.3 Overview of this thesis .............................................................................6

Chapter 2  Literature Review and Problem Analysis ..................................................9
  2.1 Defining temporal consistency ...................................................................9
  2.2 Assigning temporal constraints ...............................................................10
  2.3 Selecting checkpoints .............................................................................11
  2.4 Verifying temporal constraints ...............................................................13
  2.5 Handling temporal verification results .....................................................14
  2.6 Summary ...............................................................................................15

Chapter 3  Fundamentals .........................................................................................16
  3.1 Timed grid workflow representation .......................................................16
  3.2 Definitions of four temporal consistency states ......................................19
    3.2.1 Conventional consistency (CC) and inconsistency (CI) ...............19
    3.2.2 Limitations of CI ...........................................................................21
    3.2.3 Definitions of SC, WC, WI and SI .................................................22
    3.2.4 Further discussion of SC, WC, WI and SI ...................................25
  3.3 Summary ...............................................................................................26

Chapter 4  Assigning Fine-grained Upper Bound Constraints .............................28
  4.1 Assigning process ....................................................................................29
  4.2 Assigning algorithm .............................................................................33
4.3 Discussion and quantitative evaluation .......................................................34
4.4 Summary ....................................................................................................36

Chapter 5 Selecting Sufficient and Necessary Checkpoints for Upper Bound Constraint Verification .................................................................38
5.1 Minimum time redundancy .........................................................................39
  5.1.1 SC and WC time redundancy ...............................................................40
5.1.2 Minimum SC and WC time redundancy ...............................................42
5.1.3 Dynamic obtaining of minimum SC and WC time redundancy .........43
    5.1.3.1 Working process of DOMTR ......................................................43
    5.1.3.2 Overall analysis of DOMTR extra computation .........................50
5.2 Relationships between Minimum SC & WC Time Redundancy and SC, WC, WI & SI ......................................................................................................51
5.3 Selecting sufficient and necessary checkpoints ...........................................59
  5.3.1 Checkpoint selection along grid workflow execution .......................59
  5.3.2 Sufficiency and necessity of CSSMTR .................................................62
5.4 Comparison and quantitative evaluation .....................................................63
  5.4.1 Overall comparison .........................................................................64
  5.4.2 Quantitative evaluation ......................................................................65
    5.4.2.1 Quantitative effectiveness analysis .........................................68
    5.4.2.2 Quantitative efficiency analysis .................................................71
5.5 Summary ....................................................................................................74

Chapter 6 Dependency-enabled Upper Bound Constraint Verification ..........76
6.1 Temporal dependency between upper bound constraints .......................77
  6.1.1 SC and WC temporal dependency between two upper bound constraints ........................................................................................................77
  6.1.2 SC and WC temporal dependency between a series of upper bound constraints ........................................................................................................80
6.2 Upper bound constraint verification based on temporal dependency ........83
  6.2.1 Upper bound constraint verification at build-time stage ..................85
  6.2.2 Upper bound constraint verification at run-time execution stage .......88
6.3 Comparison and quantitative evaluation ....................................................97
Chapter 7 Applying to Lower Bound and Fixed-time Constraints

7.1 Applying to lower bound constraints

7.2 Applying to fixed-time constraints

7.2.1 SC, WC, WI and SI of fixed-time constraints

7.2.1.1 Definition of SC, WC, WI and SI

7.2.1.2 Comparing with SC, WC, WI and SI of upper bound constraints

7.2.2 Assigning fine-grained fixed-time constraints

7.2.2.1 Assigning process

7.2.2.2 Comparing with assignment of fine-grained upper bound constraints

7.2.3 Selecting sufficient and necessary checkpoints for fixed-time constraint verification

7.2.3.1 Sufficient and necessary checkpoint selection for fixed-time constraint verification

7.2.3.2 Comparing with upper bound constraints

7.2.4 Temporal dependency between fixed-time constraints

7.2.4.1 Temporal dependency between fixed-time constraints

7.2.4.2 Comparing with upper bound constraints

7.3 Summary

Chapter 8 Extension to the Handling of Temporal Verification Results

8.1 WC adjustment

8.2 Quantitative evaluation of SC, WC WI and SI

8.3 Summary

Chapter 9 Conclusions and Future Work

9.1 Summary of this thesis

9.2 Key contributions of this thesis
9.3 Future work..............................................................................................143

Bibliography........................................................................................................145

Appendix A Notation Index .................................................................................161
List of Figures

Figure 1.1 Steps in a temporal verification cycle .......................................................... 3

Figure 3.1 Definitions of CC and CI vs definitions of SC, WC, WI and SI .............. 24

Figure 4.1 Relationship between $L_k$ and $L_{M-k+1}$ .................................................. 30
Figure 4.2 A sample relationship between $U$ and $U_1$, $U_2$, ..., and $U_N$............. 31
Figure 4.3 Change trend of $C_{local}$ and $C_{global}$ by number of fine-grained upper bound constraints (i.e. $S$) ........................................................................................ 36

Figure 5.1 SC $U(a_i, a_j)$ and its SC time redundancy at $a_p$ .............................. 41
Figure 5.2 WC $U(a_k, a_l)$ and its WC time redundancy at $a_p$ .............................. 42
Figure 5.3 Sample DOMTR working process for obtaining $MTR_{SC}$ .................... 45
Figure 5.4 Relationships between minimum SC and WC time redundancy and SC, WC, WI & SI .................................................................................................................. 52
Figure 5.5 Change trend of omitted upper bound constraint verification of $CSS_{MTR}$, $CSS_1$, $CSS_2$, $CSS_3$, $CSS_4$, $CSS_5$, $CSS_6$ and $CSS_7$ ......................................................... 70
Figure 5.6 Change trend of unnecessary upper bound constraint verification of $CSS_{MTR}$, $CSS_1$, $CSS_2$, $CSS_3$, $CSS_4$, $CSS_5$, $CSS_6$ and $CSS_7$ ......................................................... 73

Figure 6.1 Non-nested upper bound constraints $U_1$ and $U_2$ ........................... 77
Figure 6.2 Nested upper bound constraints $U_1$ and $U_2$ ........................................ 78
Figure 6.3 A series of nested upper bound constraints ........................................... 81
Figure 6.4 Change trend of ineffective upper bound constraint verification computation unit number by number of inconsistent temporal dependency...100
Figure 6.5 Change trend of mean upper bound constraint verification computation
unit number by number of upper bound constraints ..............................103

Figure 7.1 Comparison between Definition 3.4 and Definition 7.1 .................107
Figure 7.2 Relationship between $F_1, F_2, \ldots, F_N$ and $F$.................................112

Figure 8.1 Change trend of handling cost difference between conventional work based on two states and our research based on four states by mean WC number (i.e. $N$) and mean WI number (i.e. $M$).................................................................135
List of Tables

Table 5.1 Omitted upper bound constraint verification of CSS_1, CSS_2, CSS_3, CSS_4, CSS_5, CSS_6 and CSS_7 on U_i (i=1, 2, ..., N) ................................................................. 68

Table 5.2 Total omitted upper bound constraint verification of CSS_1, CSS_2, CSS_3, CSS_4, CSS_5, CSS_6 and CSS_7 .................................................................................. 69

Table 5.3 Unnecessary upper bound constraint verification of CSS_1, CSS_2, CSS_3, CSS_4, CSS_5, CSS_6 and CSS_7 on U_i (i=1, 2, ..., N) ......................................................... 71

Table 5.4 Total unnecessary upper bound constraint verification of CSS_1, CSS_2, CSS_3, CSS_4, CSS_5, CSS_6 and CSS_7 ........................................................................... 72
Chapter 1

Introduction

This thesis addresses the limitations of conventional temporal verification in grid workflow systems in terms of overall temporal verification effectiveness and efficiency. The corresponding analysis is conducted for each step of the temporal verification cycle. The novel research reported in this thesis is concerned with the investigation of how to improve conventional temporal verification from the perspective of temporal verification effectiveness and efficiency. A set of new concepts, innovative methods and algorithms are developed which can promote more effective and efficient temporal verification. Comparisons, quantitative evaluations and/or mathematical proofs are also conducted at each step of the temporal verification cycle. This will demonstrate that with our new concepts, innovative methods and algorithms, we can significantly improve overall temporal verification effectiveness and efficiency.

This chapter introduces the background, motivations and key issues of this research. First, a brief introduction to temporal verification in grid workflow systems is given in Section 1.1. Then, Section 1.2 outlines the key issues of this research. Finally, Section 1.3 presents an overview of the remainder of this thesis.

1.1 Introduction to temporal verification in grid workflow systems

In grid architecture, a grid workflow system is a type of high-level grid middleware which aims to support large-scale sophisticated scientific or business processes in a
variety of complex e-science or e-business applications such as climate modelling, disaster recovery, medical surgery, high energy physics, international stock market modelling, and so on [AKMK05, ALHZ+04, BAMK+06, LG05, SKJD+04]. Such sophisticated processes often contain hundreds of thousands of computation or data intensive activities and can take a long time to complete [AKMK05, ACCE+05, SKJD+04]. These processes are modelled or redesigned as grid workflow specifications at build-time stage [AGHP06, CJSN03, FPV04, Gob04], then instantiated at run-time instantiation stage [DCLC+06, KWL02, Poc04], and finally executed at run-time execution stage. The execution is done by facilitating the super computing and data sharing ability of the underlying grid infrastructure to complete the computation or data intensive activities [Cyb05, FKNT02, Hua03, NDG06].

In reality, complex scientific or business processes are normally time constrained [AALR+04, BAV05, BBES05, MMFL+04]. Consequently, when corresponding grid workflow specifications are defined at build-time, temporal constraints are often enforced at least at the coarse-grained level [Gru04, MMFL+04, ZZDL05]. The types of temporal constraints mainly include upper bound, lower bound and fixed-time [CY05b, EPR99, EGP00]. An upper bound constraint between two activities is a relative time value so that the duration between them must be less than or equal to it [EPR99, EGP00]. A lower bound constraint between two activities is a relative time value so that the duration between them must be greater than or equal to it [EPR99, EGP00]. A fixed-time constraint at an activity is an absolute time value by which the activity must be completed [AKMK05, CY05e]. For example, a climate modelling grid workflow must be completed by a scheduled time [AKMK05], say 7:00pm, so that the weather forecasting can be broadcast at a later time on the same day. Here, 7pm is a fixed-time constraint. Some literature has also addressed two other kinds of temporal constraints: deadline and fixed-date [CY05a, EPR99, LYC04, MO99]. [MO99] divides deadline constraints into two categories: relative deadline constraints and absolute deadline constraints. A relative deadline constraint is a time value which is relative to the start time of the grid workflow. An absolute deadline constraint is an absolute time value such as 7pm, May 1. Apparently, a relative deadline constraint is an upper bound constraint whose start activity is exactly the start activity of the whole grid workflow, while an
absolute deadline constraint is just a fixed-time constraint. Hence, in this paper, we do not discuss deadline constraints separately. In addition, according to [CY05a, EPR99], a fixed-date constraint at an activity is an absolute time value by which the activity must be completed. Obviously, a fixed-date constraint is just a fixed-time constraint. So, we do not discuss fixed-date constraints separately either.

After temporal constraints are set, temporal verification is conducted at build-time, run-time instantiation and execution stages so that we can identify any temporal violations and then handle them in time in order to ensure overall temporal correctness [BWJ02, CY05e, HA00, MO99]. According to [EPR99, MO99], a temporal verification cycle generally consists of four steps as depicted in Figure 1.1.

![Figure 1.1 Steps in a temporal verification cycle](image)

Figure 1.1 Steps in a temporal verification cycle
As shown in Figure 1.1, Step 1 is to define temporal consistency. At this step, temporal consistency states are defined.

Step 2 is to assign temporal constraints. Users may first set some coarse-grained temporal constraints. Based on them, we may then assign some fine-grained ones during specific grid workflow specification and execution.

Step 3 is to select appropriate checkpoints for conducting temporal verification. A checkpoint is an activity point where we need to verify temporal constraints [CYC04, CY05b, CY05e, MO99, ZCP01]. After Steps 1 and 2, we will have the temporal consistency states and temporal constraints. Naturally, we then think to verify them. However, before doing so, we need to figure out where to conduct their verification. Verifying temporal constraints at each activity would be simple and intuitive to operate, but not efficient as we may not need to conduct temporal verification at some activities such as those which can be completed within the allowed time intervals. Therefore, we need Step 3 in the first place.

Step 4 is to verify temporal constraints. Once we select appropriate checkpoints, we then can start to verify temporal constraints to check their current consistency based on Step 1.

For each step, conventional temporal verification research and practice have presented some concepts and approaches. However, they have not paid sufficient attention to overall temporal verification effectiveness and efficiency. In the context of grid economy, resources for executing grid workflows must be used with payment as the price [ABG02, BAV05, DXP06, TLL03, WS04]. Therefore, more resources should be used for the execution of grid workflow itself rather than for temporal verification. However, poor temporal verification effectiveness or efficiency would cause more resources to be diverted into temporal verification. Consequently, temporal verification effectiveness and efficiency become an important issue which needs to be investigated in depth. In this thesis, we will systematically investigate the limitations of conventional temporal verification in terms of temporal verification effectiveness and efficiency. Based on the
investigation, we propose a set of new concepts and innovative methods and algorithms by which we can significantly improve overall temporal verification effectiveness and efficiency.

1.2 Key issues of this research

This thesis analyses the deficiencies of existing research work at each step of the temporal verification cycle in terms of temporal verification effectiveness and efficiency. The specific explanation of temporal verification effectiveness and efficiency is provided at each step. Based on this, new concepts are proposed and innovative methods and algorithms are developed so that overall temporal verification effectiveness and efficiency can be improved. Corresponding comparisons and quantitative evaluations are also conducted for demonstration purposes. Finally, to present a glimpse of handling temporal verification results so that we can achieve a more comprehensive understanding of our research, an extension is provided.

Specifically, in Step 1, we analyse the deficiencies of two temporal consistency states defined by conventional verification work in terms of temporal verification effectiveness. Then, we propose four new states so that we can achieve better temporal verification effectiveness.

In Step 2, given that a grid workflow is normally very complicated and can take a long time to complete, we argue that a number of temporal constraints are needed in terms of overall temporal verification effectiveness. Based on the analysis, we design some novel algorithms for assigning a series of fine-grained temporal constraints based on one or a few user-defined coarse-grained temporal constraints.

In Step 3, we discuss the problems of existing representative checkpoint selection strategies. In brief, they often ignore some necessary checkpoints and/or select some unnecessary ones. Ignored checkpoints would cause some temporal verification omitted, which eventually impacts overall temporal verification
effectiveness. Unnecessary checkpoints would incur some unnecessary temporal verification, which eventually impacts overall temporal verification efficiency. To overcome such limitations, we develop an innovative strategy and corresponding algorithms which only select sufficient and necessary checkpoints.

In Step 4, we move on to verification of temporal constraints. We investigate a phenomenon which is ignored by existing temporal verification work, i.e. temporal constraints are often dependent on each other in terms of their verification. We analyse the impact of temporal dependency on overall temporal verification effectiveness and efficiency. Based on this, we develop some novel temporal verification algorithms which can improve overall temporal verification effectiveness and efficiency significantly.

Finally, we present an extension to our research in this thesis about handling temporal verification results since these results are based on our four new temporal consistency states.

The significance of this research is seen in its aim to provide a set of new concepts, innovative methods and algorithms towards better overall temporal verification effectiveness and efficiency. This would eventually improve overall performance and usability of grid workflow systems because temporal verification can be viewed as a service or function of these systems. Consequently, by deploying our new concepts and innovative methods and algorithms, grid workflow systems would be able to better support large-scale sophisticated processes in complex scientific and business applications in the context of grid economy.

1.3 Overview of this thesis

In particular, this thesis deals with the design of a set of new concepts, methods and algorithms for the verification of temporal constraints so that better overall temporal verification effectiveness and efficiency can be achieved.

In Chapter 2, we analyse the research problems in detail and review major
related work for each step of the temporal verification cycle, including temporal consistency states for Step 1, fine-grained temporal constraints assignment for Step 2, checkpoint selection for Step 3 and dependency-enabled temporal verification for Step 4. We also outline the requirements for an extension to this research about the handling of temporal verification results so that a more comprehensive understanding of this research can be achieved.

In Chapter 3, we discuss the fundamentals for our research. We describe a generic timed grid workflow representation based on the directed network graph (DNG) concept [EPR99, SO00, LFZ03, LFZ04]. In addition, based on the problem analysis in Chapter 2 for Step 1 of the temporal verification cycle, we propose four new temporal consistency states aiming for better temporal verification effectiveness. We also explain why we take upper bound constraints as the major scenario to reason about our research.

Based on the problem analysis in Chapter 2 for Step 2 of the temporal verification cycle, in Chapter 4, we investigate how to assign a series of fine-grained upper bound constraints within user-defined coarse-grained ones. The assigning process is discussed in detail and an assigning algorithm is described. The corresponding comparison and quantitative evaluation are conducted to demonstrate the significant improvement of our research in this chapter on overall temporal verification effectiveness.

Based on the problem analysis in Chapter 2 for Step 3 of the temporal verification cycle, in Chapter 5, we present a novel checkpoint selection strategy which only selects sufficient and necessary checkpoints for upper bound constraint verification. Corresponding novel algorithms are described. Finally, the corresponding comparison and quantitative evaluation are conducted to demonstrate the significant improvement of our research in this chapter on overall temporal verification effectiveness and efficiency.

Based on the problem analysis in Chapter 2 for Step 4 of the temporal verification cycle, in Chapter 6, we investigate temporal dependency between upper
bound constraints and its impact on temporal verification effectiveness and efficiency. Based on this, we develop some new algorithms for upper bound constraint verification. The corresponding comparison and quantitative evaluation are conducted to demonstrate the significant improvement of our research in this chapter on overall temporal verification effectiveness and efficiency.

In Chapter 7, we investigate the applicability of corresponding research results achieved from upper bound constraints in Chapters 3, 4, 5 and 6 to lower bound and fixed-time constraints so that we can generalise our research from upper bound constraints to all temporal constraints. We can show that these results can be equally applied to lower bound constraints and can be simplified and improved for fixed-time constraints.

Chapter 8 is an extension of our research. We try to present a glimpse of the handling of temporal verification results which are based on our four new temporal consistency states. The aim is to provide a more comprehensive understanding of our research since the handling of temporal verification results is logically next step after temporal verification.

In the last chapter, namely Chapter 9, we summarise the new ideas discussed in this thesis, the major contributions of this research, and consequent further research goals.
Chapter 2

Literature Review and Problem Analysis

In this chapter, we give an overview of major related work and generally analyse their problems in terms of overall temporal verification effectiveness and efficiency. Detailed demonstrations and reasoning will be seen in the subsequent chapters. We follow the four steps of the temporal verification cycle as shown in Figure 1.1. Correspondingly, in Sections 2.1, we focus on Step 1, i.e. major related work and problem analysis in the aspect of defining temporal consistency. In Section 2.2, we focus on Step 2, i.e. major related work and problem analysis in the aspect of assigning temporal constraints. In Section 2.3, we focus on Step 3, i.e. major related work and problem analysis in the aspect of selecting checkpoints. In Section 2.4, we focus on Step 4, i.e. major related work and problem analysis in the aspect of verifying temporal constraints. Finally, in Section 2.5, we further discuss the necessity of providing an extension to this research about the handling of temporal verification results.

2.1 Defining temporal consistency

To verify temporal constraints, conventional verification work such as [EPR99, LY04, MMZ06, MO99] has defined two temporal consistency states. They are *consistency* or *inconsistency*. To distinguish them from other concepts proposed in this thesis, we denote them as CC (Conventional Consistency) and CI (Conventional
Inconsistency) respectively. Corresponding conditions are also specified in [EPR99, LY04, MMZ06, MO99]. In [EPR99, LY04, MMZ06, MO99], once a CI occurs, corresponding exception handling will be triggered to handle it. The action might be to replace the activity of grid workflow execution, adjust the grid workflow structure, compensate the completed activities, and so on [HA00, HK03, GG06, CDP06].

However, in Section 3.2, we will see that although the CC condition is reasonable, the CI condition is too restrictive as it contains several cases. Some of these cases could be handled by potential time saving of succeeding activities without triggering any exception handling [CY06d]. Some can be handled by simpler and less costly exception handling. However, in the conventional work, all different cases of CI are handled by the same exception handling. This would cause some unnecessary extra handling to those cases that do not need exception handling or can be handled by a simpler one. In real-world grid workflow systems, every exception handling often results in some cost. Especially, due to the grid workflow complexity, the handling could be very expensive [BAV05, LSKM00]. Therefore, we need to investigate how to distinguish different cases of CI so that we can achieve different temporal verification results. Then, we could deploy different handlings to treat them so as to save the handling cost towards better cost effectiveness. We view this kind of cost effectiveness as part of overall temporal verification effectiveness because it is directly based on the temporal verification results (achieved from different handlings of temporal verification results). Accordingly, overall temporal verification effectiveness would be improved.

2.2 Assigning temporal constraints

In many complex scientific and business processes such as a climate modelling process, users often set only one or a few coarse-grained temporal constraints rather than a large number of ones [AKMK05, BBES05, BAV05]. This is because a grid workflow is normally very complicated and its execution in grid environments is very dynamic [AKMK05, BBES05, BAV05]. Hence, it is very difficult for users to set a large number of temporal constraints at build-time. However, only one or a few
coarse-grained temporal constraints are not sufficient to control grid workflow execution in terms of time [CY06e]. A grid workflow normally contains hundreds of thousands of computation or data intensive activities and lasts a long time [AKMK05, ACCE+05, SKJD+04]. With only one or a few coarse-grained temporal constraints, we are not able to locally control grid workflow execution at various activity points. As a result, we might not be able to detect temporal violations and handle them in time. Or even if we can find a temporal violation, we might not be able to handle it locally as the corresponding temporal constraint might be global. This is not cost effective as global handling will affect more activities than local handling. Therefore, we need to investigate how to assign a number of fine-grained temporal constraints based on one or a few user-defined coarse-grained ones so that we can control grid workflow execution locally at various activity points. Consequently, we can detect temporal violations in time and handle them locally for better cost effectiveness. Similar to Section 2.1, we also view this kind of cost effectiveness as part of overall temporal verification effectiveness because it is directly based on assigned temporal constraints and their verification results. Accordingly, overall temporal verification effectiveness would be improved.

Some conventional work such as [San02, SKK01] has addressed the temporal constraint allocation issue. However, they mainly focus on how to figure out some overall temporal constraints by using supporting tools or methods such as Critical Path Method (CPM). They have not discussed how to assign local fine-grained temporal constraints under the condition where one or a few user-defined coarse-grained constraints have been enforced. In this thesis, we describe our effort to resolve this issue. The conventional work can be seen as a step prior to our work.

2.3 Selecting checkpoints

Once we have temporal consistency definitions and a series of temporal constraints, we can then move to verify them. At build-time and run-time instantiation stages, temporal verification is static because there are no any specific execution times. Each temporal constraint needs to be verified only once with the consideration of all
covered activities. Therefore, we need not decide at which activities we should conduct the verification. At run-time execution stage however, activity completion durations vary because the resource availability is highly dynamic [CCS05, FKT02, RYSF+03, VBW06]. Consequently, we may need to verify each temporal constraint many times at different activities. However, conducting temporal verification at every activity is not efficient as we may not have to do so at some activities such as those that can be completed within allowed time intervals. So where should we conduct temporal verification?

The activities at which we conduct temporal verification are called *checkpoints* [EPR99, MO99, ZCP01]. Some representative Checkpoint Selection Strategies (CSS) have been proposed in the literature. [EPR99] takes every activity as a checkpoint. We denote this strategy as CSS$_1$. [ZCP01] sets checkpoints at the start time and end time of each activity. We denote this strategy as CSS$_2$. [MO99] takes the start activity and each decision activity as checkpoints. We denote this strategy as CSS$_3$. [MO99] also mentions another strategy: user-defined static checkpoints. We denote this strategy as CSS$_4$. Clearly, all of CSS$_1$, CSS$_2$, CSS$_3$ and CSS$_4$ predefine checkpoints before workflow execution. We can see that CSS$_1$ and CSS$_2$ do not ignore any checkpoints as every activity is treated as a checkpoint. However, CSS$_3$ and CSS$_4$ may ignore some checkpoints as we may need to conduct temporal verification at some activities which are not defined as checkpoints. In addition, among the predefined checkpoints of CSS$_1$, CSS$_2$, CSS$_3$ and CSS$_4$, we may not need to conduct temporal verification at some of them such as those that can be completed within allowed time intervals. Therefore, CSS$_1$, CSS$_2$, CSS$_3$ and CSS$_4$ may select some unnecessary checkpoints.

Our earlier works [CYC04, CY05b, CY05e, CY06f] have attempted to improve this situation, but they still have some deficiencies. Specifically, [CYC04] selects an activity as a checkpoint when its completion duration exceeds its maximum duration. We denote this strategy as CSS$_5$. [CY05b, CY06f] select an activity as a checkpoint when its completion duration exceeds its mean duration. We denote this strategy as CSS$_6$. [CY05e] introduces a minimum proportional time redundancy for each
activity and selects an activity as a checkpoint when its completion duration is greater than its mean duration plus its minimum proportional time redundancy. We denoted this strategy as CSS7. However, in Chapter 5, we will see that CSS3 may ignore some necessary checkpoints while CSS5 and CSS7 may select some unnecessary ones.

In summary, existing representative checkpoint selection strategies often suffer from the limitations of ignoring necessary checkpoints and/or selecting unnecessary ones. Ignored checkpoints would cause some necessary temporal verification omitted, which eventually impacts overall temporal verification effectiveness. Unnecessary checkpoints would result in some unnecessary temporal verification, which eventually impacts overall temporal verification efficiency. Clearly, neither is desirable. Hence, naturally, we may ask: “Can we develop a strategy that only selects sufficient yet necessary checkpoints?” In this thesis, we answer the question positively by presenting such a strategy.

### 2.4 Verifying temporal constraints

When a series of temporal constraints are verified, they are often dependent on each other in terms of overall temporal verification effectiveness and efficiency. This is because the later verification may make the previous verification ineffective and also the later verification may utilise the previous verification results to save current verification computation for better efficiency. Therefore, the dependency between temporal constraints must be taken into consideration when temporal verification is conducted so that we can further improve overall temporal verification effectiveness and efficiency.

Some related work on temporal verification has been done which has presented some time relevant analyses and basic temporal verification methods. [AALR+04, BBES05, BGMR03, BPB06, DDPV06] analyse QoS (Quality of Service) including temporal QoS in distributed grid (workflow) systems or applications and discuss how to provide QoS. [ABG02, BAV05, TLL03, WS04] discuss grid economy issues
including the temporal aspect of grid architecture. Chen et al. [CY05b, CY05e] address the checkpoint selection issue for conducting temporal verification. [EPR99] uses a modified Critical Path Method (CPM) to conduct temporal reasoning. Their work is one of the very few projects that consider temporal reasoning at both build-time and run-time stages. [LYC04] discusses resource constraints and their verification based on some temporal information in workflow systems. They present some new resource constraint verification methods and some mechanisms for removing detected violations. [MO99] introduces minimum and maximum durations to each activity in workflow specifications. Based on this, they present a method for dynamic temporal verification. [YB05] proposes a taxonomy that characterises and classifies various workflow management approaches in grid environments. They also outline the temporal aspect in grid workflow systems.

However, the above related work does not pay sufficient attention to temporal dependency between temporal constraints and its impact on overall temporal verification effectiveness and efficiency. Accordingly, temporal constraints are verified independently of each other. Therefore, we still need to systematically investigate temporal dependency and its impact on overall temporal verification effectiveness and efficiency.

### 2.5 Handling temporal verification results

After we verify temporal constraints, we get temporal verification results. Naturally, the next logical step would be to handle these results. Apparently, this step is beyond the scope of temporal verification. The main focus of this thesis is on temporal verification. Therefore, the handling of temporal verification results will not be taken as a main stream of this thesis. However, the temporal verification results are based on our new concepts, methods and algorithms. Hence, it would be more comprehensive for understanding the research in this thesis if we can present a glimpse of how to handle temporal verification results. In this sense, we provide an extension to our research about the handling. It concentrates on the adjustment of one temporal consistency state without triggering any exception handling, and
corresponding quantitative analysis for the advantages of introducing four temporal consistency states in terms of overall temporal verification effectiveness.

Some other research and practice such as [HA00, LSKM00] have provided some concepts and approaches for handling exceptions in workflow systems. However, they have not discussed how to handle temporal verification results based on the four new temporal consistency states. Hence, the corresponding discussion still needs to be conducted.

2.6 Summary

In this chapter, the literature and problems in terms of temporal verification effectiveness and efficiency have been overviewed and analysed. At each step of a temporal verification cycle, corresponding factors which impact temporal verification effectiveness have been identified and discussed. Based on the problem analysis and literature review, requirements for new temporal consistency states, assignment of fine-grained temporal constraints, sufficient yet necessary checkpoint selection, investigation of temporal dependency, and an extension about the handling of temporal verification results have been discussed in detail.
Chapter 3

Fundamentals

In this chapter, we discuss two fundamentals for our research. One is about timed grid workflow representation in order for us to describe the work in this thesis. Based on the directed network graph (DNG) concept, we present a timed grid workflow representation with corresponding time attributes in Section 3.1. The other is about our four innovative temporal consistency states in contrast to conventional two states. They are proposed and defined in Section 3.2. Specifically, in Section 3.2.1, we summarise the definitions of conventional temporal consistency and inconsistency. In Section 3.2.2, we analyse the limitations of conventional temporal inconsistency definition in terms of its handling cost effectiveness which eventually affects the overall temporal verification effectiveness. Finally in Section 3.2.3, based on the analysis, we introduce our four temporal consistency states.

3.1 Timed grid workflow representation

To discuss temporal verification in grid workflow systems, we must represent grid workflow with certain time attributes which are expressed in some basic time units, such as minutes, hours, or days. However, the discussion of timed grid workflow models is beyond the scope of this thesis and can be found in some other references such as [AH02, AHKB03, OH05]. Rather, we describe a generic timed grid workflow representation for presenting our work.

According to [EPR99, SO00, LFZ03, LFZ04], based on the directed network
graph (DNG) concept, a grid workflow can be represented as a DNG-based grid workflow graph, where nodes correspond to activities and edges correspond to dependencies between activities. In [SO00, LFZ03, LFZ04], the iterative structure is nested in an activity that has an exit condition defined for iterative purposes. Accordingly, the corresponding DNG-based grid workflow graph is structurally acyclic. Here, we assume that the DNG-based grid workflow graph is well structured, i.e. there are no any structure errors such as deadlocks or dead activities [Aal00, WAHE06]. The structure verification is outside the scope of this thesis and can be found in some other literature such as [Aal98, Aal00, SO00, CY06a].

To represent specific time attributes, we borrow some concepts from [BM03, CP03, EPR99, MO99, ZCP01] such as activity maximum or minimum duration as a basis. For clarity and indexing purposes, the index of notations is listed in Appendix A. We denote corresponding activity time attributes as follows:

- $a_i$: the $i^{th}$ activity of a grid workflow.
- $D(a_i)$: maximum duration of $a_i$.
- $M(a_i)$: mean duration of $a_i$.
- $d(a_i)$: minimum duration of $a_i$.
- $S(a_i)$: run-time start time of $a_i$.
- $E(a_i)$: run-time end time of $a_i$.
- $R(a_i)$: run-time completion duration of $a_i$.

The mean duration $M(a_i)$ means that statistically $a_i$ can be completed within this period of time. Other time attributes are self-explanatory. According to [EPR99, MO99, ZCP01], $D(a_i)$, $M(a_i)$ and $d(a_i)$ can be obtained based on the past execution
history. The past execution history covers the delay time incurred at $a_i$ such as the setup delay, queuing delay, synchronisation delay, network latency and so on. For a specific execution of $a_i$, the delay time is included in $R(a_i)$. Normally, we have $d(a_i) \leq M(a_i) \leq D(a_i)$. If there is a path from $a_i$ to $a_j$ ($i \leq j$), we denote the maximum duration, mean duration, minimum duration and run-time real completion duration between them as follows:

- $D(a_i, a_j)$: maximum duration between $a_i$ and $a_j$.
- $M(a_i, a_j)$: mean duration between $a_i$ and $a_j$.
- $d(a_i, a_j)$: minimum duration between $a_i$ and $a_j$.
- $R(a_i, a_j)$: minimum duration between $a_i$ and $a_j$.

Normally, we have $d(a_i, a_j) \leq M(a_i, a_j) \leq D(a_i, a_j)$. For convenience, we consider one execution path in the acyclic DNG-based grid workflow graph without losing generality. As for a selective or parallel structure, each branch is an execution path. Hence, we can directly apply the results achieved in this thesis to each branch. In overall terms, for a grid workflow containing many parallel, selective and/or mixed structures, firstly, we treat each structure as an activity. Then, the whole grid workflow would be an overall execution path and we can apply the results achieved in this thesis to it. Secondly, for every structure, for each of its branches, we continue to apply these results. Thirdly, we carry out this recursive process until we complete all branches in all structures. Correspondingly, between $a_i$ and $a_j$, $D(a_i, a_j)$ is equal to the sum of all activity maximum durations, $M(a_i, a_j)$ is equal to the sum of all activity mean durations, and $d(a_i, a_j)$ is equal to the sum of all activity minimum durations.

Regarding the representation of temporal constraints, according to the description of upper bound, lower bound and fixed-time constraints in Section 1.1, conceptually a lower bound constraint is symmetrical to an upper bound constraint.
As to a fixed-time constraint, we can view the first activity of a grid workflow as its start activity. Then, the fixed-time constraint can be treated as a special case of an upper bound constraint whose start activity is the first activity of the grid workflow and whose end activity is the one at which the fixed-time constraint is. Therefore, in this thesis, we will take upper bound constraints as the major scenario to reason about our research. The corresponding results can be equally applied to lower bound constraints due to their symmetrical relationships. For fixed-time constraints, since they can be viewed as special cases of upper bound constraints, corresponding results achieved for upper bound constraints can also be applied to them. In fact, under certain circumstances, the results can be further improved. In Chapter 7, we will present a further discussion on the applicability of the results from upper bound constraints to lower bound and fixed-time constraints.

If there is an upper bound constraint between \(a_i\) and \(a_j\), we denote it and its value as follows:

- \(U(a_i, a_j)\) : upper bound constraint between \(a_i\) and \(a_j\).
- \(u(a_i, a_j)\) : value of \(U(a_i, a_j)\).

Sometimes, if there are a series of upper bound constraints, for convenience of the discussion, we may also denote them as \(U_1, U_2, U_3\) and so forth. Accordingly we denote their values as \(u(U_1), u(U_2), u(U_3)\) and so forth.

### 3.2 Definitions of four temporal consistency states

In this section, we discuss the limitations of conventional temporal inconsistency definition and present our four new temporal consistency states.

#### 3.2.1 Conventional consistency (CC) and inconsistency (CI)

As stated in Section 2.1, to verify upper bound constraints, consistency and inconsistency conditions have been defined in the conventional verification work
There are only two states for an upper bound constraint: consistency and inconsistency, i.e. an upper bound constraint is either consistent or inconsistent. In Section 2.1, we have denoted the two states as CC (Conventional Consistency) and CI (Conventional Inconsistency) respectively. Considering an upper bound constraint $U(a_i, a_j)$ between $a_i$ and $a_j$, according to the conventional verification work such as [EPR99, MO99, ZCP01], the definitions of CC and CI are summarised in Definitions 3.1, 3.2 and 3.3.

**Definition 3.1** At build-time stage, $U(a_i, a_j)$ is said to be of
1. CC if $D(a_i, a_j) \leq u(a_i, a_j)$;
2. CI if $u(a_i, a_j) < D(a_i, a_j)$.

**Definition 3.2** At run-time instantiation stage, $U(a_i, a_j)$ is said to be of
1. CC if $D(a_i, a_j) \leq u(a_i, a_j)$;
2. CI if $u(a_i, a_j) < D(a_i, a_j)$.

**Definition 3.3** At run-time execution stage, at checkpoint\(^1\) $a_p$ between $a_i$ and $a_j$, $U(a_i, a_j)$ is said to be of
1. CC if $R(a_i, a_p) + D(a_{p+1}, a_j) \leq u(a_i, a_j)$;
2. CI if $u(a_i, a_j) < R(a_i, a_p) + D(a_{p+1}, a_j)$.

We may note that Definitions 3.1 and 3.2 are similar. This is because the only different time attribute between build-time stage and run-time instantiation stage is the start time of the whole grid workflow, i.e. $S(a_j)$, but upper bound constraints are relative and consequently have nothing to do with $S(a_j)$. Meanwhile, Definition 3.3 does not cover those situations where $a_p$ is not between $a_i$ and $a_j$. This is because under such situations, the execution of $a_p$ has nothing to do with the consistency of $U(a_i, a_j)$. That is to say, there is no need to cover them in Definition 3.3.

\(^1\) Detailed discussion on checkpoints will be conducted in Chapter 5.
3.2.2 Limitations of CI

The CC conditions in Definitions 3.1, 3.2 and 3.3 are reasonable because we should try our best to keep an upper bound constraint consistent and with such conditions we can get to the maximum extent where we can ensure the consistency of an upper bound constraint. For clarity, in this research, we denote CC as SC (Strong Consistency).

Now considering the CI, we take Definition 3.3 as an example to analyse why its condition is too restrictive. The corresponding analysis for Definitions 3.1 and 3.2 is similar. In Definition 3.3, at $a_p$, if $u(a_i, a_j) < R(a_i, a_p) + D(a_{p+1}, a_j)$, $U(a_i, a_j)$ will be treated as CI and the exception handling would be triggered [EPR99, ZCP01]. However, at the run-time execution stage, the real activity completion duration varies especially because the resource availability in large-scale grid environments is highly dynamic [AR06, ET06, MA04, SGEJ+06]. As a result, there could be some time redundancy saved by the succeeding activities after $a_p$. With this time redundancy, we can derive three cases as follows:

- **Case 1:**
  Some situations of CI may be able to be adjusted to SC without triggering any exception handling as exception handling would most likely cause significant extra cost.

- **Cases 2 & 3:**
  For those situations which cannot be adjusted to SC, the specific time deficit between $u(a_i, a_j)$ and $R(a_i, a_p) + D(a_{p+1}, a_j)$ could vary depending on the current system load. It may be bigger or smaller.

  - **Case 2:**
    For the smaller one, we can trigger simpler exception handling that may adjust and compensate fewer activities and hence save more
cost than more complicated exception handling.

- Case 3:
  For the bigger one, we can leave it for more complicated exception handling which is the same as that triggered by the conventional work for all situations of CI [HA00].

In summary, it is not appropriate that the conventional work handles all situations of CI in the same way. Different situations of CI should be identified and handled differently so that corresponding handling would be more cost effective.

Based on the limitations of CI, we divide it into three different states [CY05d]. The first one may be adjusted to SC by the possible time redundancy without triggering any exception handling. The second one could be treated by simpler and more cost-saving exception handling. The third one would normally need to be treated by more complicated exception handling as that in the conventional work. Correspondingly, we divide CI into WC (Weak Consistency), WI (Weak Inconsistency) and SI (Strong Inconsistency) [CY05d]. We give the specific definitions for them and SC in Section 3.2.3.

### 3.2.3 Definitions of SC, WC, WI and SI

In this section, we give the definitions and further explanation for SC, WC, WI and SI.

**Definition 3.4** At build-time stage, $U(a_i, a_j)$ is said to be of

1. SC if $D(a_i, a_j) \leq u(a_i, a_j)$;
2. WC if $M(a_i, a_j) \leq u(a_i, a_j) < D(a_i, a_j)$;
3. WI if $d(a_i, a_j) \leq u(a_i, a_j) < M(a_i, a_j)$;
4. SI if $u(a_i, a_j) < d(a_i, a_j)$.

**Definition 3.5** At run-time instantiation stage, $U(a_i, a_j)$ is said to be of

1. SC if $D(a_i, a_j) \leq u(a_i, a_j)$;
2. WC if $M(a_i, a_j) \leq u(a_i, a_j) < D(a_i, a_j)$;
(3) WI if \( d(a_i, a_j) \leq u(a_i, a_j) < M(a_i, a_j) \);

(4) SI if \( u(a_i, a_j) < d(a_i, a_j) \).

**Definition 3.6** At run-time execution stage, at checkpoint \( a_p \) between \( a_i \) and \( a_j \), \( U(a_i, a_j) \) is said to be of

(1) SC if \( R(a_i, a_p) + D(a_{p+1}, a_j) \leq u(a_i, a_j) \);

(2) WC if \( R(a_i, a_p) + M(a_{p+1}, a_j) \leq u(a_i, a_j) < R(a_i, a_p) + D(a_{p+1}, a_j) \);

(3) WI if \( R(a_i, a_p) + d(a_{p+1}, a_j) \leq u(a_i, a_j) < R(a_i, a_p) + M(a_{p+1}, a_j) \);

(4) SI if \( u(a_i, a_j) < R(a_i, a_p) + d(a_{p+1}, a_j) \).

We may note that Definitions 3.4 and 3.5 are similar. This is due to the same reason that was stated in Section 3.2.1 for the similarity between Definitions 3.1 and 3.2. In addition, similar to Definition 3.3, in Definition 3.6 we do not cover those situations where \( a_p \) is not between \( a_i \) and \( a_j \).

For clarity, in Figure 3.1, we explicitly compare the definitions of CC and CI and the definitions of SC, WC, WI and SI.

Figure 3.1 clearly depicts the difference between CC & CI and SC, WC, WI & SI. We now further explain SC, WC, WI and SI in detail. We take run-time execution stage as an example, i.e. Scenario (c) of Figure 3.1. The corresponding explanation for build-time and run-time instantiation stages is similar. In Scenario (c), at \( a_p \), we have following four situations.

1. \( R(a_i, a_p) + D(a_{p+1}, a_j) \leq u(a_i, a_j) \):
   
   This situation means that \( U(a_i, a_j) \) can still be kept if succeeding activities can be completed by their respective maximum durations. Since activity maximum duration is carefully set and should mostly be kept, we define this state as SC. It means that in most cases \( U(a_i, a_j) \) can be kept.

23
Figure 3.1 Definitions of CC and CI vs definitions of SC, WC, WI and SI
(2) \[ R(a_i, a_p) + M(a_{p+1}, a_j) \leq u(a_i, a_j) < R(a_i, a_p) + D(a_{p+1}, a_j) \]
This situation means that, if succeeding activities take their maximum durations to complete, \( U(a_i, a_j) \) will be violated. But if they take close to their mean durations or less time to complete, \( U(a_i, a_j) \) can still be kept. Since statistically, an activity takes about its mean duration to complete, we define this state as WC. This means that with the control of succeeding activity execution based on activity mean duration, statistically \( U(a_i, a_j) \) can still be kept.

(3) \[ R(a_i, a_p) + d(a_{p+1}, a_j) \leq u(a_i, a_j) < R(a_i, a_p) + M(a_{p+1}, a_j) \]
This situation means that, if succeeding activities take their respective mean durations or more to complete, \( U(a_i, a_j) \) will be violated. However, if succeeding activities take close to their minimum durations to complete, \( U(a_i, a_j) \) may still be kept. According to the acquisition of the mean and minimum durations, this means that statistically, for most cases, \( U(a_i, a_j) \) is difficult to be kept, and only for fewer cases where all succeeding activities can be completed by their respective minimum durations, \( U(a_i, a_j) \) can be kept. Hence, we define this state as WI.

(4) \[ u(a_i, a_j) < R(a_i, a_p) + d(a_{p+1}, a_j) \]
This situation means that, even if all succeeding activities can be completed by their minimum durations, \( U(a_i, a_j) \) still cannot be kept. According to the setting of minimum durations, this means that in most cases \( U(a_i, a_j) \) cannot be kept. Therefore, we define this state as SI.

### 3.2.4 Further discussion of SC, WC, WI and SI

According to the discussion, definitions and explanation in Section 3.2.3, we need not do anything for SC as in the conventional work. Unlike the conventional work which treats WC, WI and SI by the same exception handling, we treat them differently. For WC, we should investigate how to adjust it by using the potential time redundancy without triggering any exception handling so that corresponding handling cost can be saved. For WI, compared to SI, it has a smaller time deficit to
recover to SC and there are more cases where it could be kept like SC. Therefore, we can deploy simpler and more cost-saving exception handling. For SI, we leave it for the more sophisticated and costly exception handling as in the conventional work [HA00]. Since every exception handling normally causes some cost such as compensating some completed activities [HA00], we can achieve better handling cost effectiveness with our four states than with the conventional two states. This would eventually improve the overall temporal verification effectiveness.

Apparently, along grid workflow execution, when WI or SI occurs, corresponding exception handling would be triggered to remove them. WC could be kept on the fly as it could be adjusted by potential time redundancy. Therefore, when grid workflow execution arrives at $a_i$, before the execution of $a_i$, all previous upper bound constraints would be of either SC or WC. Otherwise, the grid workflow execution would not have arrived at $a_i$ while waiting for the handling of WI and SI in the first place. Surely, after the execution of $a_i$, some of previous SC or WC upper bound constraints may change to WI or SI. Then, corresponding exception handling would be triggered again to remove them.

Further discussion of the handling for WC, WI & SI and the corresponding evaluation of better temporal verification effectiveness will be presented in Chapter 8 as this chapter focuses on providing some general fundamentals for reasoning about our research in the following chapters.

3.3 Summary

In this chapter, a generic timed grid workflow representation has been presented based on the directed network graph (DNG) concept. Denotations of corresponding time attributes have also been provided such as activity maximum, mean and minimum durations. The presentation and corresponding denotations will be used in future chapters for reasoning about temporal verification.

Moreover, a detailed analysis of the limitations of CI (Conventional
Inconsistency) has been conducted from the perspective of its handling cost effectiveness. Based on the analysis, four new temporal consistency states have been identified and defined. They are SC (Strong Consistency), WC (Weak Consistency), WI (Weak Inconsistency) and SI (Strong Inconsistency). SC corresponds to CC (Conventional Consistency) while CI has been split into WC, WI and SI. It has been briefly analysed that with the four states, we can achieve better handling cost effectiveness. As stated in Section 2.1, this cost effectiveness is part of overall temporal verification effectiveness. Therefore, we can eventually improve temporal verification effectiveness. However, this has nothing to do with how much temporal verification is conducted. Hence, temporal verification efficiency will not be improved by the four states and will be the same as that in the conventional verification work.
Chapter 4

Assigning Fine-grained Upper Bound Constraints

As discussed in Section 2.2, in a grid workflow, we often need a large number of temporal constraints so that we can control grid workflow execution at various activities. Consequently we can detect temporal violations in time and try to handle them locally rather than globally for better handling cost effectiveness. According to Section 2.2, this would improve overall temporal verification effectiveness [AKMK05, BAV05]. As explained in Section 3.1, we take upper bound constraints as the scenario to reason about our research. Correspondingly, we need a large number of upper bound constraints.

As stated in Section 2.2, users may only set one or a few coarse-grained upper bound constraints as grid workflows are normally very complicated and their execution in grid environments is very dynamic. Therefore, we need to investigate how to assign some fine-grained upper bound constraints within the coarse-grained ones at build-time stage so that we can obtain a series of upper bound constraints. This is the main objective of this chapter.

We take one of the coarse-grained upper bound constraints as an example to discuss how to assign fine-grained upper bound constraints within its timeframe at build-time stage. The corresponding results can also be equally applied to each of other coarse-grained upper bound constraints. Overall, there may be overlapping assignments. Once this occurs, we take the assignment with the biggest value.
Specifically, in Section 4.1, we investigate the assigning process. In Section 4.2, we present the overall assigning algorithm. In Section 4.3, we conduct an overall quantitative analysis to show how better handling cost effectiveness can be achieved with fine-grained upper bound constraints.

4.1 Assigning process

We denote a concerned coarse-grained upper bound constraint as $U$ and its value as $u(U)$. We suppose that $U$ cover $T$ activities. For simplicity, we number these activities from $a_1$, i.e. they are $a_1, a_2, ..., a_T$. Meanwhile, as indicated in Section 3.2.4, when grid workflow execution arrives at an activity, before the execution of the activity, all previous upper bound constraints would be of either SC or WC. We will only focus on SC in this chapter. The corresponding discussion for WC is similar, hence omitted. Accordingly, we can suppose that $U$ be of SC.

Within $U$, based on the past execution history, we can summarise those grid workflow path slots where temporal violations often happen. Accordingly, at each of these slots we should set a fine-grained upper bound constraint. We suppose there be $N$ such grid workflow path slots. Correspondingly, we need to set $N$ fine-grained upper bound constraints. We denote them as $U_1, U_2, ..., U_N$, and their values as $u(U_1), u(U_2), ..., u(U_N)$. We suppose $U_i$ cover $M_i$ activities, denoted as $a_{ij}$ ($j=1, 2, 3, ..., M_i$). Among all of $U_i$ ($i = 1, 2, 3, ..., N$), there may be some fine-grained upper bound constraints which cover some activities in common. Since $U$ is of SC, according to item 1 of Definition 3.4, we have a time redundancy: $u(U) - D(a_1, a_T)$. This time redundancy can be used to balance certain time deviation incurred by abnormal grid workflow execution. Based on the time redundancy, we can derive $U_1, U_2, ..., U_N$ as follows.

Suppose there be $M$ activities in total covered by $U_1, U_2, ..., U_N$. Note that $M$ may not be equal to $M_1 + M_2 + M_3 + ... + M_N$ because some of $U_1, U_2, ..., U_N$ may have some activities in common. Among $M$ activities, we first sort all $D(a_s) - M(a_s)$ ($s = 1, 2, 3, ..., M$) in ascending order to get a sorting list. We denote the list
as $L$ and the items in $L$ as $L_1, L_2, \ldots, L_M$. We also denote the numbers of activities corresponding to $L_1, L_2, \ldots, L_M$ as $l_1, l_2, \ldots, l_M$. If $D(a_i) - M(a_i)$ is ranked number $k$ in $L$, i.e. $L_k$, then we propose formula (4-1) below to allocate $u(U) - D(a_i, a_T)$ to each of $M$ activities [CY06e]. We denote the time quota allocated to $a_i$ as $TQ(a_i)$.

$$TQ(a_i) = [u(U) - D(a_i, a_T)] \frac{L_{M-k+1}}{\sum_{i=l_i}^{M} [D(a_i) - M(a_i)]} \quad (1 \leq k \leq M)$$  \hspace{1cm} (4-1)

The relationship between $L_k$ and $L_{M-k+1}$ is depicted in Figure 4.1.

![Figure 4.1 Relationship between $L_k$ and $L_{M-k+1}$](image)

We now further explain formula (4-1). In formula (4-1), we allocate $u(U) - D(a_i, a_T)$ to the activities covered by $U_1, U_2, \ldots, U_N$ based on the difference between activity maximum duration and activity mean duration. The activity with a bigger difference will be allocated a smaller quota of $u(U) - D(a_i, a_T)$. This is because statistically, an activity will be completed around its mean duration. Therefore, the activity with a bigger difference between its maximum duration and its mean duration has more redundant time to compensate the possible time deviation incurred by abnormal grid workflow execution. Hence, we should allocate a smaller quota to it.

After we allocate $u(U) - D(a_i, a_T)$ to the activities covered by $U_1, U_2, \ldots, U_N$, each activity $a_{ij}$ ($i = 1, 2, 3, \ldots, N; j = 1, 2, 3, \ldots, M_i$) will carry a time quota. We can
then derive the values of $U_1$, $U_2$, ..., and $U_N$. Considering $U_i$, we derive its value by formula (4-2) below.

$$u(U_i) = \sum_{j=1}^{M_i} \left[ TQ(a_{ij}) + D(a_{ij}) \right] \quad (i=1, 2, 3, ..., N) \quad (4-2)$$

A sample relationship between $U$ and $U_1$, $U_2$, ..., and $U_N$ is depicted in Figure 4.2.

![Figure 4.2 A sample relationship between $U$ and $U_1$, $U_2$, ..., and $U_N$](image)

We now further explain formula (4-2). Considering $a_{ij}$ covered by $U_i$, $TQ(a_{ij})$ is the time quota allocated to it and $D(a_{ij})$ is its maximum duration. Apparently, as long as $a_{ij}$ can be completed within $TQ(a_{ij}) + D(a_{ij})$, the execution of $a_{ij}$ is normal and will not the impact temporal correctness of overall grid workflow execution. Correspondingly, for all $M_i$ activities covered by $U_i$ together, i.e. $a_{ij} (j = 1, 2, 3, ..., M_i)$, we can see: if all $M_i$ activities can be completed within $\sum_{j=1}^{M_i} \left[ TQ(a_{ij}) + D(a_{ij}) \right]$,

then their execution will be normal and will not impact the temporal correctness of overall grid workflow execution. Therefore, we set the value of $u(U_i)$ to $\sum_{j=1}^{M_i} \left[ TQ(a_{ij}) + D(a_{ij}) \right]$, i.e. as shown in formula (4-2).
To demonstrate the applicability of formulas (4-1) and (4-2), we must prove that all assigned fine-grained upper bound constraints are also of SC. Otherwise, the assigning process may cause some new temporal violations and hence should not be deployed. We derive Theorem 4.1 to support the applicability.

**Theorem 4.1**
Let $U$ be of SC. If we allocate its time redundancy $u(U) - D(a_1, a_T)$ according to formula (4-1) and assign fine-grained upper bound constraints according to formula (4-2), then, if $U$ is of SC, all fine-grained upper bound constraints are also of SC.

**Proof:**
Considering a fine-grained upper bound constraint, say $U_i$, if $U$ is of SC, then $u(U) \geq D(a_1, a_T)$, i.e. $u(U) - D(a_1, a_T) \geq 0$. Meanwhile, according to formula (4-1), $TQ(a_{ij})$ is a share of $u(U) - D(a_1, a_T)$. Hence, we have: $TQ(a_{ij}) \geq 0$ ($j=1, 2, 3, \ldots, M_i$). With formula (4-2), we have: $u(U_i) = \sum_{j=1}^{M_i} [TQ(a_{ij}) + D(a_j)] \geq \sum_{j=1}^{M_i} D(a_j) = D(a_{i1}, a_{iM_i})$, i.e. we have inequation (4-3) below.

$$u(U_i) \geq D(a_{i1}, a_{iM_i}) \quad (4-3)$$

According to item 1 of Definition 3.4, inequation (4-3) means that $U_i$ is of SC.

Thus, in overall terms, the theorem holds.

We may note that the above assigning process would cause some extra computation. However, from formulas (4-1) and (4-2), we can see that basically the extra computation is only one or two additions, multiplications or divisions at each activity covered by corresponding time slots. From Chapter 6, we will see that we need to conduct a large amount of verification computation for verifying upper bound constraints many times at various activities [MO99, ZCP01]. Compared to this, the one or two additions, multiplications or divisions at each activity would be negligible.
### 4.2 Assigning algorithm

Based on the discussion of Section 4.1, we can derive an algorithm for assigning fine-grained upper bound constraints at build-time stage. The main part of the algorithm is depicted in Algorithm 4.1.

| Input       | ArrayUA: an array of all \( T \) activities covered by \( U \);                        |
|            | ArrayMA: an array of all \( M \) activities covered by all time slots where fine-grained upper bound constraints need to be assigned; |
|            | Maximum and mean durations of all activities in ArrayUA and ArrayMA;              |
|            | \( U \): a coarse-grained upper bound constraint                                   |
| Output     | Fine-grained upper bound constraints within \( U \).                                |
| Step 1     | Sorting all \( D(a_s) - M(a_s) \) of \( M \) activities in ArrayMA              |
|            | 1.1 For each activity from ArrayMA, say \( a_s \), compute \( D(a_s) - M(a_s) \); |
|            | 1.2 Sort all \( D(a_s) - M(a_s) \) in ascending order to ArrayLA. Suppose \( D(a_s) - M(a_s) \) be ranked No. \( k \) in ArrayLA; |
| Step 2     | Allocating the time redundancy of the upper bound constraint \( U \), i.e. \( u(U) - D(a_i, a_T) \), to all \( M \) activities in ArrayMA; |
|            | 2.1. Based on ArrayUA, compute \( u(U) - D(a_i, a_T) \);                         |
|            | 2.2. For each of \( M \) activities in ArrayMA, say \( a_s \) (\( s = 1, 2, 3, \ldots, M \)), compute its time quota \( TQ(a_s) \) as follows, i.e. formula (4-1); |

\[
TQ(a_s) = \left[ u(U) - D(a_i, a_T) \right] \frac{L_{M-k+1}}{\sum_{l=l_i}^{L_{M-k+1}} [D(a_T) - M(a_l)]} \quad (1 \leq k \leq M)
\]
Step 3: Computing values of fine-grained upper bound constraints

3.1 For each of fine-grained upper bound constraints, say $U_i$ at the $i^{th}$ time slot, we compute its value as follows, i.e. formula (4-2);

$$u(U_i) = \sum_{j=1}^{M_i} [TQ(a_{ij}) + D(a_{ij})] \quad (i=1, 2, 3, \ldots, N)$$

Algorithm 4.1 Assigning fine-grained upper bound constraints at build-time stage

4.3 Discussion and quantitative evaluation

With fine-grained upper bound constraints, we can control grid workflow execution locally at various activities. When a temporal violation happens, we can try to handle it locally rather than globally within $U$. Local handling would affect fewer activities and hence is more cost effective than global handling. Now, we conduct a quantitative analysis so that we can have a clear picture of how the introduction of fine-grained upper bound constraints can achieve better cost effectiveness.

Within $U$, we suppose there be $S$ fine-grained upper bound constraints. For a fine-grained upper bound constraint, we suppose that statistically there be $X$ temporal violations which can be handled within it. For simplicity, we assume that the number of activities between any two adjacent fine-grained upper bound constraints be the same, denoted as $Q$, and each fine-grained upper bound constraint cover the same number of activities, denoted as $P$. In addition, we also assume that $X$ temporal violations happen respectively at the first $X$ activities of each fine-grained upper bound constraint. We denote the exception handling cost for an activity as $C$. We denote the exception handling cost based on $U$ as $C_{global}$, and that based on fine-grained upper bound constraints as $C_{local}$. Then, the improvement on overall cost effectiveness is reflected by how $C_{local}$ is less than $C_{global}$.
For the $k^{th}$ temporal violation in the $i^{th}$ fine-grained upper bound constraint, we can derive that the exception handling cost based on $U$ is $[i*Q+(i-1)*P+k]*C$, and that the exception handling cost based on fine-grained upper bound constraints is $k*C$. Therefore, for $S$ fine-grained upper bound constraints in total, we have formulas (4-17) and (4-18) below.

\[
C_{local} = S*\sum_{k=1}^{X} k*C
\]  
(4-17)

\[
C_{global} = \sum_{i=1}^{S} \sum_{k=1}^{X} ([i*Q+(i-1)*P+k]*C)
\]  
(4-18)

We now take a set of specific values to see how formulas (4-17) and (4-18) perform. We suppose that $P=3$, $Q=2$, $X=2$, $C$ be equal to 1 cost unit. We also suppose that $S$ can change from 0 to 20. The selection of these specific values is rather random and does not affect our analysis because what we want to see is the trend of how $C_{local}$ and $C_{global}$ change to $S$. With $S$ changing, we list corresponding $C_{local}$ and $C_{global}$ in Figure 4.3.

According to Figure 4.3, we can see that with $S$ increasing, both $C_{local}$ and $C_{global}$ are increasing. However, their increase rates are quite different. $C_{local}$ increases slowly while $C_{global}$ increases dramatically. Particularly, as $S$ gets larger, $C_{global}$ becomes much greater than $C_{local}$. As stated in Section 2.2, in real-world grid workflow systems, a grid workflow normally contains hundreds of thousands of activities and takes a long time to complete [AKMK05, SKJD+04, YBT05]. Consequently, to better control local grid workflow execution locally at various activities, a number of fine-grained upper bound constraints are often needed. That is to say, in real-world grid workflow systems, $S$ is normally a large number. Therefore, in overall terms, we can conclude that introducing a series of fine-grained upper bound constraints can achieve much better cost effectiveness.
4.4 Summary

Having one or a few user-defined coarse-grained upper bound constraints is simple and easy for users. However, it is not sufficient to control and monitor grid workflow execution locally at various activities. Consequently, we might not be able to detect temporal violations in time and handle them locally for better handling cost effectiveness.

In this chapter, we have investigated how to assign fine-grained upper bound constraints within user-defined coarse-grained ones so that we can obtain a series of
upper bound constraints. Corresponding build-time assigning process has been discussed in detail and its assigning algorithm has been developed. A quantitative evaluation has been conducted to demonstrate the significant improvement by fine-grained upper bound constraints on cost effectiveness in handling temporal violations. As stated in Section 2.2, this kind of cost effectiveness is part of overall temporal verification effectiveness. Therefore, the research outcomes described in this chapter would eventually contribute to the improvement of temporal verification effectiveness significantly.

Regarding temporal verification efficiency, although the assigning process would cause some extra computation, as explained at the end of Section 4.1, such extra computation would be negligible. That is to say, temporal verification efficiency would not be impacted much and would remain about the same as that in the conventional verification work.

Once we have assigned a series of fine-grained upper bound constraints, by nature, we will then verify them. However, before we can conduct the verification, we must find out where this should be done. This issue is known as checkpoint selection which will be investigated in Chapter 5. The temporal verification will be discussed later in Chapter 6.
Chapter 5

Selecting Sufficient and Necessary Checkpoints for Upper Bound Constraint Verification

Once a series of upper bound constraints have been assigned or set, their verification must be conducted so that we can detect any temporal violations and handle them in time. As stated in Section 2.3, at build-time and run-time instantiation stages, upper bound constraint verification is static and we need not to be concerned about where to conduct the verification. At run-time execution stage however, we need to select appropriate checkpoints for conducting the verification. In Section 2.3, we have outlined seven existing representative checkpoint selection strategies: CSS1, CSS2, CSS3, CSS4, CSS5 and CSS7. We have also analysed the problem of CSS1, CSS2, CSS3, CSS4. That is that they often ignore some necessary checkpoints and/or select some unnecessary ones. In Section 5.4, we will see that our preliminary strategies CSS5, CSS6 and CSS7 also have the similar problem.

As described in Section 2.3, ignored checkpoints would cause some necessary temporal verification omitted, which would eventually impact overall temporal verification effectiveness. Unnecessary checkpoints would incur some unnecessary temporal verification, which would eventually impact overall temporal verification efficiency. Apparently, neither is desirable. Therefore, to overcome such limitations of existing representative checkpoint selection strategies for better overall temporal verification effectiveness and efficiency, in this chapter, we develop a novel strategy
that can select only sufficient yet necessary checkpoints dynamically along grid workflow execution.

Specifically, in Section 5.1, we introduce the new concept of minimum time redundancy which will serve as a key reference parameter for our strategy. We also develop a method for dynamically obtaining the minimum time redundancy along grid workflow execution. After that, in Section 5.2, we investigate the relationships between minimum time redundancy and upper bound constraint consistency. Based on these relationships, in Section 5.3, we present our new strategy and rigorously prove its sufficiency and necessity for checkpoint selection. In Section 5.4, we conduct a comprehensive comparison and quantitative evaluation to demonstrate that our strategy can significantly improve temporal verification effectiveness and efficiency over the existing representative strategies.

### 5.1 Minimum time redundancy

As indicated in Section 3.2.4, when grid workflow execution arrives at an activity, before the execution of the activity, all previous upper bound constraints would be of either SC or WC. Therefore, checkpoint selection actually focuses on selecting checkpoints for verifying previous SC and WC upper bound constraints to check their current consistency. Accordingly, minimum time redundancy consists of minimum SC and WC time redundancy [CY07]. The former is for SC upper bound constraints and the later is for WC upper bound constraints.

First, we introduce the concept of SC and WC time redundancy for one upper bound constraint in Section 5.1.1. Then, we introduce minimum SC and WC time redundancy for multiple upper bound constraints in Section 5.1.2. After that, in Section 5.1.3, we discuss how to obtain minimum SC and WC time redundancy dynamically along grid workflow execution.
5.1.1 SC and WC time redundancy

At run-time execution stage, we consider a SC upper bound constraint, say \( U(a_i, a_j) \), as shown in Figure 5.1. At activity point \( a_p \) between \( a_i \) and \( a_j \), according to item 1 of Definition 3.6, we have inequation (5-1) below.

\[
R(a_i, a_p) + D(a_{p+1}, a_j) \leq u(a_i, a_j)
\]  

(5-1)

Clearly, there is a time difference which is \( u(a_i, a_j) - [R(a_i, a_p) + D(a_{p+1}, a_j)] \). This difference indicates that if the execution of succeeding activities can be controlled within the difference, \( U(a_i, a_j) \) can still be kept as SC even if the execution consumes more time than scheduled. Correspondingly, we define this time difference as SC time redundancy of \( U(a_i, a_j) \) at activity point \( a_p \), and denote it as \( TR_{SC}(U(a_i, a_j), a_p) \) (\( TR_{SC} \): SC Time Redundancy). Accordingly, we have Definition 5.1 below.

**Definition 5.1 (SC Time Redundancy)**

At activity point \( a_p \) between \( a_i \) and \( a_j \) (i<j), let \( U(a_i, a_j) \) be of SC. Then, SC time redundancy of \( U(a_i, a_j) \) at \( a_p \) is defined as \( u(a_i, a_j) - [R(a_i, a_p) + D(a_{p+1}, a_j)] \) and denoted as \( TR_{SC}(U(a_i, a_j), a_p) \), i.e.

\[
TR_{SC}(U(a_i, a_j), a_p) = u(a_i, a_j) - [R(a_i, a_p) + D(a_{p+1}, a_j)]
\]  

(5-2)

For a WC upper bound constraint, say \( U(a_k, a_l) \) as shown in Figure 5.2, according to item 2 of Definition 3.6, we have inequation (5-3) below.

\[
R(a_k, a_p) + M(a_{p+1}, a_l) \leq u(a_k, a_l)
\]  

(5-3)

Similarly, there is a time difference that is \( u(a_k, a_l) - [R(a_k, a_p) + M(a_{p+1}, a_l)] \). This difference indicates that if the execution of succeeding activities can be controlled within the difference, \( U(a_k, a_l) \) can still be kept as WC. We define this time difference as WC time redundancy of \( U(a_k, a_l) \) at activity point \( a_p \), and denote it as \( TR_{WC}(U(a_k, a_l), a_p) \) (\( TR_{WC} \): WC Time Redundancy). Accordingly, we have
Definition 5.2 below.

**Definition 5.2 (WC Time Redundancy)**

At activity point \( a_p \) between \( a_k \) and \( a_l \) \((k<l)\), let \( U(a_k, a_l) \) be of WC. Then, WC time redundancy of \( U(a_k, a_l) \) at \( a_p \) is defined as \( u(a_k, a_l) - [R(a_k, a_p) + M(a_{p+1}, a_l)] \) and denoted as \( TR_{WC}(U(a_k, a_l), a_p) \), i.e.

\[
TR_{WC}(U(a_k, a_l), a_p) = u(a_k, a_l) - [R(a_k, a_p) + M(a_{p+1}, a_l)]
\]  (5-4)

For clarity, we further depict \( TR_{SC}(U(a_i, a_j), a_p) \) and \( TR_{WC}(U(a_k, a_l), a_p) \) in Figures 5.1 and 5.2 explicitly.

**Figure 5.1 SC \( U(a_i, a_j) \) and its SC time redundancy at \( a_p \)**
5.1.2 Minimum SC and WC time redundancy

We now consider multiple SC or WC upper bound constraints that cover $a_p$. Based on Definitions 5.1 and 5.2, we introduce minimum SC and WC time redundancy in Definitions 5.3 and 5.4 respectively.

**Definition 5.3 (Minimum SC Time Redundancy)**

Let $U_1, U_2, \ldots, U_N$ be $N$ SC upper bound constraints and all of them cover $a_p$. Then, at $a_p$, minimum SC time redundancy is defined as the minimum one of all SC time redundancies of $U_1, U_2, \ldots, U_N$, and is denoted as $MTR_{SC}(a_p)$ ($MTR_{SC}$: SC Minimum Time Redundancy).

$$MTR_{SC}(a_p) = \min \{ TR_{SC}(U_s, a_p) | s = 1, 2, \ldots, N \} \quad (5-5)$$

**Definition 5.4 (Minimum WC Time Redundancy)**

Let $U_1, U_2, \ldots, U_N$ be $N$ WC upper bound constraints and all of them cover $a_p$. Then, at $a_p$, minimum WC time redundancy is defined as the minimum one of all WC time redundancies of $U_1, U_2, \ldots, U_N$, and is denoted as $MTR_{WC}(a_p)$ ($MTR_{WC}$: WC Minimum Time Redundancy).

$$MTR_{WC}(a_p) = \min \{ TR_{WC}(U_s, a_p) | s = 1, 2, \ldots, N \} \quad (5-5)$$
redundancies of $U_1, U_2, \ldots, U_N$, and is denoted as $MTR_{WC}(a_p)$ \textbf{(MTR$_{WC}$: WC Minimum Time Redundancy)}.

\[ MTR_{WC}(a_p) = \min \{ TR_{WC}(U_s, a_p) | s = 1, 2, \ldots, N \} \quad (5-6) \]

Apparently, according to Definitions 5.3 and 5.4, at $a_{p-1}$ or just before the execution of $a_p$, minimum SC and WC time redundancies are $MTR_{SC}(a_{p-1})$ and $MTR_{WC}(a_{p-1})$ respectively.

In addition, we normally have $M(a_p) + MTR_{WC}(a_{p-1}) < D(a_p) + MTR_{SC}(a_{p-1})$. The reason is simple: if $M(a_p) + MTR_{WC}(a_{p-1}) \geq D(a_p) + MTR_{SC}(a_{p-1})$, then, since the upper bound constraint of $MTR_{SC}(a_{p-1})$ is of SC, according to items 1 and 2 of Definition 3.6, the upper bound constraint of $MTR_{WC}(a_{p-1})$ must also be of SC; however, the upper bound constraint of $MTR_{WC}(a_{p-1})$ is of WC.

\subsection*{5.1.3 Dynamic obtaining of minimum SC and WC time redundancy}

Along grid workflow execution, at $a_p$, an intuitive method for obtaining $MTR_{SC}(a_p)$ and $MTR_{WC}(a_p)$ is to compute and compare all SC and WC time redundancies. However, this method is inefficient as it causes too much extra computation. Therefore, we develop a more efficient method and denote it as DOMTR (Dynamic Obtaining of Minimum Time Redundancy). DOMTR dynamically obtains $MTR_{SC}$ and $MTR_{WC}$ along grid workflow execution by utilising corresponding verification computation results of upper bound constraints. We now describe in detail its working process in Section 5.1.3.1. In Section 5.1.3.2, we address the overall extra computation of DOMTR.

\subsubsection*{5.1.3.1 Working process of DOMTR}

DOMTR works at run-time instantiation and execution stages. At run-time instantiation stage, it sets up some initial values. Then, at run-time execution stage, it dynamically obtains $MTR_{SC}$ and $MTR_{WC}$ along grid workflow execution. To immediately provide a clear picture of the DOMTR working process, we first present a sample DOMTR working process in Figure 5.3 for obtaining $MTR_{SC}$. The
corresponding sample process for obtaining $MTR_{WC}$ is similar. Note that Figure 5.3 is only for providing an immediate picture. The detailed discussion is presented later.

From Figure 5.3, we can see that DOMTR initialises some values at run-time instantiation stage. For example, it sets $MTR_{SC}$ to the biggest possible float number of the system for each activity which is not covered by any SC upper bound constraints. At run-time instantiation stage, we can see that there are three types of key activity points for computing $MTR_{SC}$. We list them below. Different computation needs to be conducted respectively at the three types of activity points.

- Start activity points of one or multiple upper bound constraints.

- Intermediate activity points covered by one or multiple upper bound constraints.

- End activity points of one or multiple upper bound constraints.

At run-time instantiation stage:

1) For each SC upper bound constraint, say $U(a, a)$, compute its time difference $u(a, a) - D(a, a)$, denoted as SC time difference.

2) At the start activity of each SC upper bound constraint, say $a$, compare all SC time differences to get the minimum one, denoted as $SMTD_{SC-init}(a)$ ($SMTD$: Minimum Time Difference at Start activity; $init$: initial).

3) At the end activity of each SC upper bound constraint, say $a$, compare all SC time differences of those SC upper bound constraints which cover $a$, but do not end at $a$ to get the minimum one, denoted as $EMTD_{SC-init}(a)$ ($EMTD$: Minimum Time Difference at End activity; $init$: initial).

4) Set $MTR_{SC}$ to the biggest float number of the system for each activity which is not covered by any SC upper bound constraints.

(a) At run-time instantiation stage (setting up initial values)
At run-time execution stage:

\[ MTR_{SC}(a_j) = \text{the biggest possible float number of the system} \]

\[ MTR_{SC}(a_j) = EMTD_{SC-out}(a_j) - \sum_{i=0}^{n} [R(a_i) - D(a_i)] \]

\[ MTR_{SC}(a_{\psi}) = \min \{SMDT_{SC-out}(a_{\psi}), MTR_{SC}(a_{\psi})\} - [R(a_{\psi}) - D(a_{\psi})] \]

\[ MTR_{SC}(a_j) = MTR_{SC}(a_{\psi}) - [R(a_{\psi}) - D(a_{\psi})] \]

\[ MTR_{SC}(a_j) = SMDT_{SC-out}(a_j) - [R(a_j) - D(a_j)] \]

\[ MTR_{SC}(a_j) = \text{the biggest possible float number of the system} \]

(b) At run-time execution stage (obtaining \( MTR_{SC} \) along grid workflow execution)

Figure 5.3 Sample DOMTR working process for obtaining \( MTR_{SC} \)
Based on the sample working process in Figure 5.3, we now list the complete working process of DOMTR.

At run-time instantiation stage (setting up initial values)

Step 1.
During the upper bound constraint verification process, for each SC upper bound constraint, say $U(a_i, a_j)$, compute a time difference $u(a_i, a_j) - D(a_i, a_j)$, denoted as SC time difference. For each WC upper bound constraint, say $U(a_k, a_l)$, compute another time difference $u(a_k, a_l) - M(a_k, a_l)$, denoted as WC time difference. All such time differences can be obtained by using corresponding computation results from upper bound constraint verification. This is because $D(a_i, a_j)$ and $M(a_k, a_l)$ have been computed when the verification of $U(a_i, a_j)$ and $U(a_k, a_l)$ is conducted at build-time stage.

Step 2.
At every start activity of each SC upper bound constraint, derive minimum SC time difference. For example, at $a_i$ of SC $U(a_i, a_j)$, compare all SC time differences of all SC upper bound constraints that cover $a_i$ to derive the minimum one. Denote the minimum one as $SMTD_{SC-init}(a_i)$ ($SMTD$: Minimum Time Difference at Start activity; init: initial). At every start activity of each WC upper bound constraint, derive minimum WC time difference. For example, at $a_k$ of WC $U(a_k, a_l)$, compare all WC time differences of all WC upper bound constraints that cover $a_k$ to derive the minimum one. Denote the minimum one as $SMTD_{WC-init}(a_k)$.

Step 3.
At every end activity of each SC upper bound constraint, derive another minimum SC time difference. But this one is different from the one mentioned in Step 2. For example, at $a_j$ of SC $U(a_i, a_j)$, compare all SC time differences of those SC upper bound constraints which cover $a_j$, but do not end at $a_j$. Denote the minimum one as $EMTD_{SC-init}(a_j)$ ($EMTD$: Minimum Time Difference at End).

\[\text{Details of temporal verification would be discussed in Chapter 6.}\]
activity; *init:* initial). At every end activity of each WC upper bound constraint, derive another minimum WC time difference. For example, at $a_l$ of WC $U(a_k, a_l)$, compare all WC time differences of all WC upper bound constraints which cover $a_l$, but do not end at $a_l$. Denote the minimum one as $EMTD_{WC-init}(a_l)$.

**Step 4.**
For each activity, say $a_r$, which is not covered by any SC or WC upper bound constraints, set $MTR_{SC}(a_r)$ and $MTR_{WC}(a_r)$ to the biggest possible float number of the system (denoted as BFN) which is far greater than any $SMTD_{SC-init}$ and $SMTD_{WC-init}$.

**At run-time execution stage** (computing $MTR_{SC}$ and $MTR_{WC}$)

**Step 5.**
Along grid workflow execution, suppose now the execution arrives at a start activity of some SC and/or WC upper bound constraints, say $a_i$. There are three cases:

- **Case 1:**
  $a_i$ is the start activity of some SC and WC upper bound constraints.

- **Case 2:**
  $a_i$ is the start activity of some SC upper bound constraints only.

- **Case 3:**
  $a_i$ is the start activity of some WC upper bound constraints only.

**Step 5.1 (Case 1)**
For $MTR_{SC}(a_i)$:
If $SMTD_{SC-init}(a_i) < MTR_{SC}(a_{i-1})$, then
$$MTR_{SC}(a_i) = SMTD_{SC-init}(a_i) - [R(a_i) - D(a_i)]$$
Else
$$MTR_{SC}(a_i) = MTR_{SC}(a_{i-1}) - [R(a_i) - D(a_i)]$$
End if

For $MTR_{WC}(a_i)$:
If $SMTD_{WC-init}(a_i) < MTR_{WC}(a_{i-1})$, then
   \[ MTR_{WC}(a_i) = SMTD_{WC-init}(a_i) - [R(a_i) - M(a_i)] \]
Else
   \[ MTR_{WC}(a_i) = MTR_{WC}(a_{i-1}) - [R(a_i) - M(a_i)] \]
End if

Step 5.2 (Case 2)
For $MTR_{SC}(a_i)$:
If $SMTD_{SC-init}(a_i) < MTR_{SC}(a_{i-1})$, then
   \[ MTR_{SC}(a_i) = SMTD_{SC-init}(a_i) - [R(a_i) - D(a_i)] \]
Else
   \[ MTR_{SC}(a_i) = MTR_{SC}(a_{i-1}) - [R(a_i) - D(a_i)] \]
End if

For $MTR_{WC}(a_i)$:
If $MTR_{WC}(a_{i-1}) = BFN$, then
   \[ MTR_{WC}(a_i) = BFN \]
Else
   \[ MTR_{WC}(a_i) = MTR_{WC}(a_{i-1}) - [R(a_i) - M(a_i)] \]
End if

Step 5.3 (Case 3)
For $MTR_{SC}(a_i)$:
If $MTR_{SC}(a_{i-1}) = BFN$, then
   \[ MTR_{SC}(a_i) = BFN \]
Else
   \[ MTR_{SC}(a_i) = MTR_{SC}(a_{i-1}) - [R(a_i) - D(a_i)] \]
End if

For $MTR_{WC}(a_i)$:
If $SMTD_{WC\text{-init}}(a_i) < MTR_{WC}(a_{i-1})$, then

$$MTR_{WC}(a_i) = SMTD_{WC\text{-init}}(a_i) - [R(a_i) - M(a_i)]$$

Else

$$MTR_{WC}(a_i) = MTR_{WC}(a_{i-1}) - [R(a_i) - M(a_i)]$$

End if

**Step 6.**

Along grid workflow execution, suppose now the execution arrives at an activity, say $a_p$. $a_p$ is covered by some SC or WC upper bound constraints, but is neither a start activity nor an end activity of any SC or WC upper bound constraints. After the execution of $a_p$,

$$MTR_{SC}(a_p) = MTR_{SC}(a_{p-1}) - [R(a_p) - D(a_p)]$$

$$MTR_{WC}(a_p) = MTR_{WC}(a_{p-1}) - [R(a_p) - M(a_p)].$$

**Step 7.**

Along grid workflow execution, we now discuss how to obtain new $MTR_{SC}$ and $MTR_{WC}$ when grid workflow execution arrives at the end activity of some SC and/or WC upper bound constraints. We discuss how to obtain new $MTR_{SC}$ only. For new $MTR_{WC}$, the corresponding discussion is similar. Suppose now grid workflow execution arrives at the end activity $a_j$ of some SC upper bound constraints. Denote the SC upper bound constraint corresponding to $MTR_{SC}(a_{j-1})$ as $U(MTR_{SC}(a_{j-1}))$.

**Step 7.1.**

If $a_j$ is not the end activity of $U(MTR_{SC}(a_{j-1}))$, obtain $MTR_{SC}(a_j)$ according to Step 6 as $U(MTR_{SC}(a_{j-1}))$ is still valid.

**Step 7.2.**

If $a_j$ is the end activity of $U(MTR_{SC}(a_{j-1}))$, then, $MTR_{SC}(a_j)$ can also be obtained according to Step 6. However, such $MTR_{SC}(a_j)$ cannot be used further after the execution of $a_j$ because $U(MTR_{SC}(a_{j-1}))$ will be invalid. For example, we cannot
compute $MTR_{SC}(a_{j+1})$ based on such $MTR_{SC}(a_j)$. Therefore, after the execution of $a_j$, we need to compute new $MTR_{SC}(a_j)$ to replace such $MTR_{SC}(a_j)$. The new $MTR_{SC}(a_j)$ depends on two cases:

- **Case 1:**
  There are no other SC upper bound constraints which cover $a_j$ but do not end at $a_j$.

- **Case 2:**
  There are some other SC upper bound constraints which cover $a_j$ but do not end at $a_j$.

**Step 7.2.1 (Case 1)**
The new $MTR_{SC}(a_j)$ is set to BFN.

**Step 7.2.2 (Case 2)**
Suppose that the upper bound constraint corresponding to $EMTD_{SC\cdot init}(a_j)$ is $U(a_m, a_n) \ (m \leq j < n)$. Then, the new $MTR_{SC}(a_j) = EMTD_{SC\cdot init}(a_j) - \sum_{s=m}^{j} [R(a_s) - D(a_s)]$.

**Step 8.**
When grid workflow execution arrives at an activity which is not covered by any SC or WC upper bound constraints, do nothing and simply keep the initial values set by Step 4.

**Step 9.**
Along grid workflow execution, repeat all or some of Steps 5, 6, 7 and 8 when applicable.

**5.1.3.2 Overall analysis of DOMTR extra computation**

Compared to the intuitive method which needs to compute and compare all SC and WC time redundancies at each activity to obtain $MTR_{SC}$ and $MTR_{WC}$ of that activity,
DOMTR involves far less extra computation.

At run-time instantiation stage, DOMTR sets up some initial values by directly using the corresponding computation results from upper bound constraint verification at build-time stage. According to Chapter 6, upper bound constraint verification must be conducted at build-time stage regardless of whether or not we select some checkpoints for the verification at run-time execution stage. Hence, DOMTR does not incur any extra computation.

At run-time execution stage, DOMTR computes minimum SC and WC time redundancy on the fly along grid workflow execution. From the working steps of DOMTR listed in Section 5.1.3.1, we can see that basically the extra computation we need is only one or two subtractions or comparisons at each activity covered by one or more upper bound constraints. Compared to a large amount of verification computation which is needed for verifying a number of upper bound constraints many times at various activities [MO99, ZCP01], such one or two subtractions or comparisons would be negligible.

5.2 Relationships between Minimum SC & WC Time Redundancy and SC, WC, WI & SI

At run-time execution stage, at activity point $a_p$, we discuss the relationships between $MTR_{SC}(a_{p-1})$ & $MTR_{WC}(a_{p-1})$ and SC, WC, WI & SI. We first depict these relationships in Figure 5.4 to immediately present an overall picture. Then, we establish them in Theorems 5.1, 5.2 and 5.3.
Figure 5.4 Relationships between minimum SC and WC time redundancy and SC, WC, WI & SI

Theorem 5.1
At activity point $a_p$, if $R(a_p) > D(a_p) + MTR_{SC}(a_{p-1})$, then:
1) all previous WC upper bound constraints now cannot be of SC and may be of WC, WI or SI;
2) the previous SC upper bound constraint whose minimum SC time redundancy at $a_{p-1}$ is $MTR_{SC}(a_{p-1})$ now cannot be of SC and may be of WC,
WI or SI; and
3) all other previous SC upper bound constraints may now be of SC, WC, WI or SI.

**Proof:**

1) We suppose that $U(a_k, a_l)$ be a previous WC upper bound constraint, i.e. it is of WC before execution of $a_p$. We also suppose that $a_p$ be between $a_k$ and $a_l$. Then, according to item 2 of Definition 3.6, we have inequation (5-7) below.

\[ u(a_k, a_l) < R(a_k, a_{p-1}) + D(a_p, a_l) \] (5-7)

If $R(a_p) > D(a_p) + MTR_{SC}(a_{p-1})$, we have inequation (5-8) below.

\[ D(a_p) < R(a_p) \] (5-8)

Combining inequations (5-7) and (5-8) together, we have: $u(a_k, a_l) < R(a_k, a_{p-1}) + D(a_p, a_l) = R(a_k, a_{p-1}) + D(a_p) + D(a_{p+1}, a_l) < R(a_k, a_{p-1}) + R(a_p) + D(a_{p+1}, a_l) = R(a_k, a_p) + D(a_{p+1}, a_l)$. Hence, we have inequation (5-9) below.

\[ u(a_k, a_l) < R(a_k, a_p) + D(a_{p+1}, a_l) \] (5-9)

According to item 1 of Definition 3.6, inequation (5-9) means that $U(a_k, a_l)$ can not be of SC after execution of $a_p$.

In addition, from $R(a_p) > D(a_p) + MTR_{SC}(a_{p-1})$, we have: $R(a_k, a_{p-1}) + M(a_p, a_l) = R(a_k, a_{p-1}) + M(a_p) + M(a_{p+1}, a_l) \leq R(a_k, a_{p-1}) + D(a_p) + M(a_{p+1}, a_l) < R(a_k, a_{p-1}) + R(a_p) - MTR_{SC}(a_{p-1}) + M(a_{p+1}, a_l) = R(a_k, a_p) + M(a_{p+1}, a_l) - MTR_{SC}(a_{p-1})$. Hence, we have inequation (5-10) below.

\[ R(a_k, a_{p-1}) + M(a_p, a_l) < R(a_k, a_p) + M(a_{p+1}, a_l) - MTR_{SC}(a_{p-1}) \] (5-10)

Meanwhile, because $U(a_k, a_l)$ is previously of WC, according to item 2 of Definition 3.6, we have inequation (5-11) below.
\[ R(a_k, a_{p-1}) + M(a_{p-1}, a_i) \leq u(a_k, a_i) \quad (5-11) \]

However, from inequations (5-10) and (5-11), we cannot judge whether inequation (5-12) below holds or not.

\[ R(a_k, a_p) + M(a_{p+1}, a_i) \leq u(a_k, a_i) \quad (5-12) \]

If inequation (5-12) holds, according to item 2 of Definition 3.6, \( U(a_k, a_i) \) is of WC again. However, depending on specific \( MTR_{SC}(a_{p-1}) \), inequation (5-12) may or may not hold. Similarly, we may or may not have inequation (5-13) below.

\[ R(a_k, a_p) + d(a_{p+1}, a_i) \leq u(a_k, a_i) < R(a_k, a_p) + M(a_{p+1}, a_i) \quad (5-13) \]

Meanwhile, we also may or may not have inequation (5-14) below.

\[ u(a_k, a_i) < R(a_k, a_p) + d(a_{p+1}, a_i) \quad (5-14) \]

Therefore, according to items 2, 3 and 4 of Definition 3.6, depending on specific \( MTR_{SC}(a_{p-1}) \), after execution of \( a_p \), \( U(a_k, a_i) \) may be of WC, WI or SI.

2) We suppose that the previous SC upper bound constraint corresponding to \( MTR_{SC}(a_{p-1}) \) be \( U(a_i, a_j) \). Then, according to Definitions 5.1 and 5.3, we have inequation (5-15) below.

\[ MTR_{SC}(a_{p-1}) = u(a_i, a_j) - [R(a_i, a_{p-1}) + D(a_{p}, a_j)] \quad (5-15) \]

With \( R(a_p) > D(a_p) + MTR_{SC}(a_{p-1}) \), we have: \( u(a_i, a_j) - [R(a_i, a_{p-1}) + D(a_{p}, a_j)] < R(a_p) - D(a_p) \), i.e. \( u(a_i, a_j) < R(a_p) - D(a_p) + [R(a_i, a_{p-1}) + D(a_{p}, a_j)] = R(a_i, a_p) + D(a_{p+1}, a_j) \). Hence, we have inequation (5-16) below.

\[ u(a_i, a_j) < R(a_i, a_p) + D(a_{p+1}, a_j) \quad (5-16) \]
According to item 1 of Definition 3.6, inequation (5-16) means that $U(a_i, a_j)$ cannot be of SC after execution of $a_p$.

In addition, Similar to 1), depending on specific $MTR_{SC}(a_{p-1})$, we may or may not have inequations (5-17), (5-18) and (5-19) below.

$$R(a_i, a_p) + M(a_{p+1}, a_j) \leq u(a_i, a_j) < R(a_i, a_p) + D(a_{p+1}, a_j) \quad (5-17)$$

$$R(a_i, a_p) + d(a_{p+1}, a_j) \leq u(a_i, a_j) < R(a_i, a_p) + M(a_{p+1}, a_j) \quad (5-18)$$

$$u(a_i, a_j) < R(a_i, a_p) + d(a_{p+1}, a_j) \quad (5-19)$$

Therefore, according to items 2, 3 and 4 of Definition 3.6, depending on specific $MTR_{SC}(a_{p-1})$, after execution of $a_p$, $U(a_i, a_j)$ may be of WC, WI or SI.

3) We suppose that $U(a_i, a_j)$ be a previous SC upper bound constraint, i.e. it is of SC before execution of $a_p$. We also suppose that $a_p$ be between $a_i$ and $a_j$. In addition, we suppose that $U(a_i, a_j)$ be not the one whose minimum SC time redundancy at $a_{p-1}$ is $MTR_{SC}(a_{p-1})$. According to Definitions 5.1 and 5.3, we have inequation (5-20) below.

$$MTR_{SC}(a_{p-1}) \leq TR_{SC}(U(a_i, a_j), a_p) \quad (5-20)$$

Together with $R(a_p) > D(a_p) + MTR_{SC}(a_{p-1})$, it is insufficient to decide whether inequation (5-21) below holds or not.

$$TR_{SC}(U(a_i, a_j), a_p) + D(a_p) < R(a_p) \quad (5-21)$$

If inequation (5-21) holds, then, similar to 2), we can prove that $U(a_i, a_j)$ cannot be of SC after execution of $a_p$. If inequation (5-21) does not hold, i.e. inequation (5-22) below holds, then, similar to 1), we can derive that depending on specific $TR_{SC}(U(a_i, a_j), a_p)$, we may or may not have inequation (5-23).
\[ R(a_p) \leq TRSC(U(a_i, a_j), a_p) + D(a_p) \] (5-22)

\[ R(a_i, a_p) + D(a_{p+1}, a_j) \leq u(a_i, a_j) \] (5-23)

In addition, we also may or may not have inequations (5-17), (5-18) or (5-19). Therefore, according to items 1, 2, 3 and 4 of Definition 3.6, depending on specific TRSC(U(a_i, a_j), a_p), after execution of a_p, U(a_i, a_j) may be of SC, WC, WI or SI.

Thus, in overall terms, the theorem holds. □

**Theorem 5.2**
At activity point a_p, if \( M(a_p) + MTR_{WC}(a_{p-1}) < R(a_p) \leq D(a_p) + MTR_{SC}(a_{p-1}) \), then:
1) all previous SC upper bound constraints are now still of SC;
2) the previous WC upper bound constraint whose minimum WC time redundancy at \( a_{p-1} \) is \( MTR_{WC}(a_{p-1}) \) now cannot be of WC and SC, and may now be of WI or SI; and
3) all other previous WC upper bound constraints may now be of SC, WC, WI or SI.

**Proof:**
1) We suppose that \( U(a_i, a_j) \) be of SC before execution of a_p. According to item 1 of Definition 3.6, we have \( R(a_i, a_{p-1}) + D(a_p, a_j) \leq u(a_i, a_j) \). If \( R(a_p) \leq D(a_p) + MTR_{SC}(a_{p-1}) \), then \( R(a_i, a_p) + D(a_{p+1}, a_j) = R(a_i, a_{p-1}) + R(a_p) + D(a_{p+1}, a_j) \leq R(a_i, a_{p-1}) + MTR_{SC}(a_{p-1}) + D(a_p) + D(a_{p+1}, a_j) = R(a_i, a_{p-1}) + MTR_{SC}(a_{p-1}) + D(a_p, a_j) \leq R(a_i, a_{p-1}) + TRSC(U(a_i, a_j), a_{p-1}) + D(a_p, a_j) = u(a_i, a_j) \). Hence, we have inequation (5-24) below.

\[ R(a_i, a_p) + D(a_{p+1}, a_j) \leq u(a_i, a_j) \] (5-24)

According to item 1 of Definition 3.6, inequation (5-24) means that \( U(a_i, a_j) \) is still of SC after execution of a_p.
2) The proof is similar to 2) of Theorem 5.1, hence omitted.

3) The proof is similar to 3) of Theorem 5.1, hence omitted.

Thus, in overall terms, the theorem holds. □

Theorem 5.3
At activity point $a_p$, if $R(a_p) \leq M(a_p) + MTR_{WC}(a_{p-1})$, then:

1) all previous SC upper bound constraints are now still of SC;
2) all previous WC upper bound constraints are now at least of WC and may be of SC; and
3) if previous WC upper bound constraints are now still of WC, their status has been moved closer to SC.

Proof:
1) The proof is similar to 1) of Theorem 5.2, hence omitted.

2) We suppose that $U(a_k, a_l)$ be of WC before execution of $a_p$. According to item 2 of Definition 3.6, we have inequation (5-25) below.

$$R(a_k, a_{p-1}) + M(a_p, a_l) \leq u(a_k, a_l) < R(a_k, a_{p-1}) + D(a_p, a_l)$$  (5-25)

If $R(a_p) \leq MTR_{WC}(a_{p-1}) + M(a_p)$, then we have: $R(a_k, a_p) + M(a_{p+1}, a_l) = R(a_k, a_{p-1}) + R(a_p) + M(a_{p+1}, a_l) \leq R(a_k, a_{p-1}) + MTR_{WC}(a_{p-1}) + M(a_p) + M(a_{p+1}, a_l) = R(a_k, a_{p-1}) + MTR_{WC}(a_{p-1}) + M(a_{p+1}, a_l) \leq R(a_k, a_{p-1}) + TR_{WC}(U(a_k, a_l), a_{p-1}) + M(a_p, a_l) = u(a_k, a_l)$. Hence, we have inequation (5-26) below.

$$R(a_k, a_p) + M(a_{p+1}, a_l) \leq u(a_k, a_l)$$  (5-26)

In addition, we also have: $R(a_k, a_p) + D(a_{p+1}, a_l) = R(a_k, a_{p-1}) + R(a_p) + D(a_{p+1}, a_l) \leq R(a_k, a_{p-1}) + MTR_{WC}(a_{p-1}) + M(a_p) + D(a_{p+1}, a_l) \leq R(a_k, a_{p-1}) + MTR_{WC}(a_{p-1}) + D(a_p, a_l)$. Hence, we have inequation (5-27) below.
$R(a_k, a_p) + D(a_{p+1}, a_l) \leq R(a_k, a_{p-1}) + MTR_{WC}(a_{p-1})$  

(5-27)

However, from inequations (5-26) and (5-27), we cannot judge whether inequation (5-28) below holds or not.

$$u(a_k, a_l) < R(a_k, a_p) + D(a_{p+1}, a_l)$$  

(5-28)

In fact, depending on specific $MTR_{WC}(a_{p-1})$, inequation (5-28) may or may not hold. If inequation (5-28) holds, then, combining it with inequation (5-26), we have inequation (5-29) below.

$$R(a_k, a_p) + M(a_{p+1}, a_l) \leq u(a_k, a_l) < R(a_k, a_p) + D(a_{p+1}, a_l)$$  

(5-29)

According to item 2 of Definition 3.6, inequation (5-29) means that $U(a_k, a_l)$ is of WC. If inequation (5-28) does not hold, then, we have inequation (5-30) below.

$$R(a_k, a_p) + D(a_{p+1}, a_l) \leq u(a_k, a_l)$$  

(5-30)

According to item 1 of Definition 3.6, inequation (5-30) means that $U(a_k, a_l)$ already switches to SC after execution of $a_p$.

3) If $R(a_p) \leq MTR_{WC}(a_{p-1}) + M(a_p)$, then we have inequation (5-31) below.

$$R(a_p) \leq TR_{WC}(U(a_k, a_l), a_{p-1}) + M(a_p)$$  

(5-31)

Hence, we have: $R(a_k, a_p) + M(a_{p+1}, a_l) = R(a_k, a_{p-1}) + R(a_p) + M(a_{p+1}, a_l) \leq R(a_k, a_{p-1}) + TR_{WC}(U(a_k, a_l), a_{p-1}) + M(a_p) + M(a_{p+1}, a_l)$. Therefore, we have inequation (5-32) below.

$$R(a_k, a_p) + M(a_{p+1}, a_l) \leq R(a_k, a_{p-1}) + M(a_p, a_l) + TR_{WC}(U(a_k, a_l), a_{p-1})$$  

(5-32)

Correspondingly, we have inequation (5-33) below.
\[ u(a_k, a_l) - [R(a_k, a_{p-1}) + M(a_{p}, a_l) + \mathcal{TR}_{WC}(U(a_k, a_l), a_{p-1})] \leq u(a_k, a_l) - [R(a_k, a_p) + M(a_{p+1}, a_l)] \quad (5-33) \]

Inequation (5-33) means that, after execution of \( a_p \), \( U(a_k, a_l) \) is closer to SC than before.

Thus, in overall terms, the theorem holds. \( \square \)

5.3 Selecting sufficient and necessary checkpoints

In this section, we present our new checkpoint selection strategy based on the relationships between minimum SC & WC time redundancy and SC, WC, WI and SI. This strategy selects sufficient yet necessary checkpoints on the fly along grid workflow execution. We first derive its selection process in Section 5.3.1. Then in Section 5.3.2, we rigorously prove its sufficiency and necessity for checkpoint selection.

5.3.1 Checkpoint selection along grid workflow execution

According to Figure 5.4 and Section 5.2, at \( a_p \), we can draw the following three conclusions:

- **Conclusion 1 for** \( R(a_p) > D(a_p) + \mathcal{MTR}_{SC}(a_{p-1}) \):  
  We need to verify all previous SC and WC upper bound constraints. There is at least one previous SC upper bound constraint which is violated and now is not of SC. It is exactly the one whose SC time redundancy at \( a_{p-1} \) is \( \mathcal{MTR}_{SC}(a_{p-1}) \).

- **Conclusion 2 for** \( M(a_p) + \mathcal{MTR}_{WC}(a_{p-1}) < R(a_p) \leq D(a_p) + \mathcal{MTR}_{SC}(a_{p-1}) \):  
  Only previous WC upper bound constraints need to be verified but not the SC constraints. And there is at least one previous WC upper bound constraint
which is violated and now is not of SC and WC. It is exactly the one whose WC time redundancy at \( a_{p-1} \) is \( MTR_{WC}(a_{p-1}) \).

- **Conclusion 3 for** \( R(a_p) \leq M(a_p) + MTR_{WC}(a_{p-1}) \):
  
  We need not verify all previous SC upper bound constraints. As to previous WC upper bound constraints, according to Theorem 5.3, after the execution of \( a_p \), their status has been moved closer to SC (sometimes can even be changed to SC). Therefore, if a previous WC upper bound constraint is still of WC after the execution of \( a_p \), the previous handling or adjustment on it can be carried forward. Hence, we need not do anything further to it. That is to say, we need not verify it.

Based on the above three conclusions, we can decide on the fly whether we should take \( a_p \) as a checkpoint when grid workflow execution arrives at \( a_p \). We denote the decision-making approach as \( CDA(a_p) \) (Checkpoint Decision-making Approach at \( a_p \)) and describe its decision process as follows.

At activity \( a_p \):

- If \( R(a_p) > D(a_p) + MTR_{SC}(a_{p-1}) \), we take it as a checkpoint for verifying SC, WC, WI & SI of all previous SC upper bound constraints, and for verifying WC, WI & SI of all previous WC upper bound constraints;
- If \( M(a_p) + MTR_{WC}(a_{p-1}) < R(a_p) \leq D(a_p) + MTR_{SC}(a_{p-1}) \), we take \( a_p \) as a checkpoint for verifying SC, WC, WI & SI of all previous WC upper bound constraints only;
- If \( R(a_p) \leq M(a_p) + MTR_{WC}(a_{p-1}) \), we do not take \( a_p \) as a checkpoint.

Combining \( CDA(a_p) \) with DOMTR from Section 5.1.3, we can derive a novel checkpoint selection strategy that dynamically selects not only sufficient but also necessary checkpoints along grid workflow execution. We denote the strategy as \( CSS_{MTR} \) (Minimum Time Redundancy based Checkpoint Selection Strategy). The overall selection process of \( CSS_{MTR} \) is:

Along grid workflow execution, \( CSS_{MTR} \) calls DOMTR to compute the
minimum SC and WC time redundancy of each activity. Then when grid workflow execution arrives at an activity, say $a_p$, $CSS_{MTR}$ calls $CDA(a_p)$ to decide whether $a_p$ should be taken as a checkpoint.

The main part of $CSS_{MTR}$ is depicted in Algorithm 1 below.

<table>
<thead>
<tr>
<th>Input</th>
<th>Maximum, minimum and mean durations of all activities; all SC and WC upper bound constraints.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>$True$ or $False$ as an appropriate checkpoint.</td>
</tr>
<tr>
<td>Step 1</td>
<td>At run-time instantiation stage, conduct DOMTR to set up some initial values based on the computation results of upper bound constraint verification.</td>
</tr>
<tr>
<td>1.1.</td>
<td>Execute Steps 1 and 2 of DOMTR to obtain all $SMTD_{SC-init}$ and $SMTD_{WC-init}$.</td>
</tr>
<tr>
<td>1.2.</td>
<td>Execute Step 3 of DOMTR to obtain $EMTD_{SC-init}$ and $EMTD_{WC-init}$ for every end activity of each SC or WC upper bound constraint.</td>
</tr>
<tr>
<td>1.3.</td>
<td>Execute Step 4 of DOMTR to set the biggest possible float number of the system to $MTR_{SC}$ and $MTR_{WC}$ of each activity that is not covered by any SC or WC upper bound constraints.</td>
</tr>
<tr>
<td>Step 2</td>
<td>At run-time execution stage, conduct DOMTR (refer to Section 5.1.3.1) to obtain $MTR_{SC}(a_{p-1})$ and $MTR_{WC}(a_{p-1})$ when grid workflow execution arrives at $a_{p-1}$.</td>
</tr>
</tbody>
</table>
2.1 If \( a_{p-1} \) is a start activity of some SC and/or WC upper bound constraints, then execute Step 5 of DOMTR including Steps 5.1, 5.2 and 5.3 to obtain \( MTR_{SC}(a_{p-1}) \) and \( MTR_{WC}(a_{p-1}) \).

2.2 If \( a_{p-1} \) is an intermediate activity of some SC and/or WC upper bound constraints, then execute Step 6 of DOMTR to obtain \( MTR_{SC}(a_{p-1}) \) and \( MTR_{WC}(a_{p-1}) \).

2.3 If \( a_{p-1} \) is an end activity of some SC and/or WC upper bound constraints, then execute Step 7 of DOMTR including Steps 7.1, 7.2, 7.2.1, 7.2.2 to obtain \( MTR_{SC}(a_{p-1}) \) and \( MTR_{WC}(a_{p-1}) \).

2.4 If \( a_{p-1} \) is not covered by any SC or WC upper bound constraints, then execute Step 8 of DOMTR to obtain \( MTR_{SC}(a_{p-1}) \) and \( MTR_{WC}(a_{p-1}) \).

### Algorithm 5.1 Checkpoint selection process of CSS\(_{MTR}\)

#### 3.1 Execute Steps 1 and 2 of DOMTR to obtain all \( SMTD_{SC-init} \) and \( SMTD_{WC-init} \).

#### 3.2 Call \( CDA(a_p) \) to compare \( D(a_p) + MTR_{SC}(a_{p-1}) \) and \( M(a_p) + MTR_{WC}(a_{p-1}) \) with \( R(a_p) \) so that we can decide whether \( a_p \) should be selected as an appropriate checkpoint.

#### 3.3 According to 3.1, output “True” or “False” for selecting \( a_p \) as a checkpoint.

### 5.3.2 Sufficiency and necessity of CSS\(_{MTR}\)

We now further prove that checkpoints selected by \( CSS_{MTR} \) along grid workflow execution are both sufficient and necessary for upper bound constraint verification.
Theorem 5.4 (Sufficiency)
Along grid workflow execution, the checkpoints selected by $CSS_{MTR}$ are sufficient for upper bound constraint verification, namely no checkpoints are ignored.

**Proof:**
With $CSS_{MTR}$, at $a_p$, we consider whether we should take it as a checkpoint only if $M(a_p) + MTR_{WC}(a_{p-1}) < R(a_p)$. In fact, according to the discussion in Sections 5.2 and 5.3.1, we need not take $a_p$ as a checkpoint if $M(a_p) + MTR_{WC}(a_{p-1}) \geq R(a_p)$. Therefore, the checkpoints selected by $CSS_{MTR}$ are sufficient, i.e. none are ignored.

Thus, in overall terms, the theorem holds. □

Theorem 5.5 (Necessity)
Along grid workflow execution, all checkpoints selected by $CSS_{MTR}$ are necessary for upper bound constraint verification, i.e. there are no any unnecessary checkpoints.

**Proof:**
According to Figure 5.4 and the three conclusions in Section 5.3.1, we can see that once we take an activity, say $a_p$, as a checkpoint, there must be at least one WC or SC upper bound constraint which will be violated. It is exactly the one whose minimum WC or SC time redundancy at $a_{p-1}$ is just $MTR_{WC}(a_{p-1})$ or $MTR_{SC}(a_{p-1})$. That is to say, taking $a_p$ as a checkpoint is necessary.

Thus, in overall terms, the theorem holds. □

5.4 Comparison and quantitative evaluation

In this section, we evaluate our checkpoint selection strategy $CSS_{MTR}$ by comparing it with other representative strategies, namely $CSS_1$, $CSS_2$, $CSS_3$, $CSS_4$ as well as our initial strategies, namely $CSS_5$, $CSS_6$ and $CSS_7$. These strategies are mentioned in
Section 2.3.

5.4.1 Overall comparison

According to Theorems 5.4 and 5.5, all checkpoints selected by CSS_MTR along grid workflow execution are both sufficient and necessary. Therefore, CSS_MTR does not omit any necessary upper bound constraint verification or incur any unnecessary verification. Moreover, according to Section 5.1.3.2, the extra computation caused by DOMTR would be negligible. From Algorithm 5.1, we can see that DOMTR is actually the key component of CSS_MTR. Therefore, the extra computation caused by conducting CSS_MTR would also be negligible.

According to Section 2.3, CSS_1 and CSS_2 do not ignore any checkpoints instead they set checkpoints at every activity, which is very inefficient. CSS_1 and CSS_2 often ignore some necessary ones. Meanwhile, CSS_1, CSS_2, CSS_3 and CSS_4 may select some unnecessary checkpoints. Therefore, CSS_MTR is more effective than CSS_3 and CSS_4 for upper bound constraint verification and is also more efficient than CSS_1, CSS_2, CSS_3 and CSS_4.

According to [CYC04], CSS_5 takes an activity, say a_p, as a checkpoint if D(a_p) < R(a_p). Compared to CSS_MTR which takes a_p as a checkpoint when M(a_p) + MTR_{WC}(a_p-1) < R(a_p) ≤ D(a_p). That is to say, CSS_5 may ignore some necessary checkpoints. Therefore, CSS_MTR is more effective than CSS_5 for upper bound constraint verification.

According to [CY05b], CSS_6 takes an activity, say a_p, as a checkpoint if M(a_p) < R(a_p). Compared to CSS_MTR which takes a_p as a checkpoint when M(a_p) + MTR_{WC}(a_p-1) < R(a_p), the situation is unnecessary where M(a_p) < R(a_p) ≤ M(a_p) + MTR_{WC}(a_p-1). That is to say, CSS_6 may select some unnecessary checkpoints. Therefore, CSS_MTR is more efficient than CSS_6 for upper bound constraint verification.

According to [CY05e], CSS_7 introduces minimum proportional WC time redundancy to a_p-1, denoted as MPTR_{WC}(a_p-1). Then, at a_p, CSS_7 takes it as a
checkpoint if \( M(a_p) + MPTR_{WC}(a_{p-1}) < R(a_p) \). In [CY05e], the authors first compute all time redundancies of all upper bound constraints. Then, they allocate each time redundancy to every involved activity in certain proportion. After that, \( MPTR_{WC}(a_{p-1}) \) is equal to the minimum quota of \( a_{p-1} \). However, according to Section 5.1.2, \( MTR_{WC}(a_{p-1}) \) is equal to the minimum time redundancy rather than its proportional share. That is to say, \( MPTR_{WC}(a_{p-1}) \) is actually part of \( MTR_{WC}(a_{p-1}) \). Therefore, we have inequation (5-34) below.

\[
MPTR_{WC}(a_{p-1}) \leq MTR_{WC}(a_{p-1})
\]  

(5-34)

Based on inequation (5-34), we can find out the unnecessary situation for CSS7 to take an activity, say \( a_p \) as a checkpoint. Comparing the condition for CSS7 to take \( a_p \) as a checkpoint, i.e. \( M(a_p) + MPTR_{WC}(a_{p-1}) < R(a_p) \), with the condition for CSS\(_{MTR}\) to take \( a_p \) as a checkpoint, we can see that the unnecessary situation where CSS7 takes \( a_p \) as a checkpoint is \( M(a_p) + MPTR_{WC}(a_{p-1}) < R(a_p) \leq M(a_p) + MTR_{WC}(a_{p-1}) \). This means that CSS7 may also select some unnecessary checkpoints. Therefore, CSS\(_{MTR}\) is more efficient than CSS7 for upper bound constraint verification.

### 5.4.2 Quantitative evaluation

We now conduct a quantitative analysis so that we can provide a clear picture of how our CSS\(_{MTR}\) is more effective and efficient than CSS\(_1\), CSS\(_2\), CSS\(_3\), CSS\(_4\), CSS\(_5\), CSS\(_6\) and CSS\(_7\). Correspondingly, we compare their respective omitted and unnecessary upper bound constraint verification. According to the temporal consistency definitions in Section 3.2.3, the primary verification computation is focused on summing up maximum durations between two activities. Therefore, we take each maximum duration addition operation as a verification computation unit. Correspondingly, we analyse CSS\(_1\), CSS\(_2\), CSS\(_3\), CSS\(_4\), CSS\(_5\), CSS\(_6\), CSS\(_7\) and CSS\(_{MTR}\) by comparing their numbers of omitted and unnecessary verification computation units. For this task, we make the following denotations.

- \( O_{css_i} \): number of omitted verification computation units of CSS\(_i\).
• $O_{css_2}$ : number of omitted verification computation units of $CSS_2$.

• $O_{css_3}$ : number of omitted verification computation units of $CSS_3$.

• $O_{css_4}$ : number of omitted verification computation units of $CSS_4$.

• $O_{css_5}$ : number of omitted verification computation units of $CSS_5$.

• $O_{css_6}$ : number of omitted verification computation units of $CSS_6$.

• $O_{css_7}$ : number of omitted verification computation units of $CSS_7$.

• $O_{css_{MTR}}$ : number of omitted verification computation units of $CSS_{MTR}$.

• $U_{css_1}$ : number of unnecessary verification computation units of $CSS_1$.

• $U_{css_2}$ : number of unnecessary verification computation units of $CSS_2$.

• $U_{css_3}$ : number of unnecessary verification computation units of $CSS_3$.

• $U_{css_4}$ : number of unnecessary verification computation units of $CSS_4$.

• $U_{css_5}$ : number of unnecessary verification computation units of $CSS_5$.

• $U_{css_6}$ : number of unnecessary verification computation units of $CSS_6$.

• $U_{css_7}$ : number of unnecessary verification computation units of $CSS_7$. 

66
\begin{itemize}
  \item $U_{cssMTR}$: number of unnecessary verification computation units of $CSS_{MTR}$.
  
  We consider $N$ WC upper bound constraints in a complex grid workflow, denoted as $U_1$, $U_2$, \ldots, $U_N$. The corresponding discussion for SC upper bound constraints is similar. We consider the most complicated situation where the $N$ WC upper bound constraints are nested one after another. We introduce the following temporary variables.
  
  \begin{itemize}
    \item $X$: number of activities covered by $U_1$.
    
    \item $Y$: number of activities between two adjacent upper bound constraints.
    
    \item $L$: number of start activity and all decision activities concerned to $CSS_3$.
    
    \item $M$: number of predefined checkpoints by $CSS_4$.
    
    \item $Q_1$: possibility for $R(a_p) \leq D(a_p)$.
    
    \item $Q_2$: possibility for $R(a_p) \leq M(a_p) + MTR_{WC}(a_{p-1})$.
    
    \item $Q_3$: possibility for $R(a_p) \leq M(a_p) + MPTR_{WC}(a_{p-1})$.
    
    \item $Q_4$: possibility for $R(a_p) \leq M(a_p)$.
  \end{itemize}

  In these variables, for simplicity, we assume that $Y$ is the same for any two adjacent upper bound constraints. We also assume that all activities have the same $Q_1$, $Q_2$, $Q_3$ and $Q_4$. In addition, we suppose that all $L$ and $M$ activities be covered by $U_1$.

  Apparently, we have $0 \leq Q_1, Q_2, Q_3, Q_4 \leq 1$. According to Section 5.4.1 and Definition 5.4, we have $Q_1 \geq Q_2 \geq Q_3 \geq Q_4$. Obviously, we have:
• $Q_1 - Q_2$ : possibility for $M(a_p) + MTR_{WC}(a_{p-1}) < R(a_p) \leq D(a_p)$.

• $Q_2 - Q_3$ : possibility for $M(a_p) + MPTR_{WC}(a_{p-1}) < R(a_p) \leq M(a_p) + MTR_{WC}(a_{p-1})$.

• $Q_2 - Q_4$ : possibility for $M(a_p) < R(a_p) \leq M(a_p) + MTR_{WC}(a_{p-1})$.

Then, for $CSS_{MTR}$, according to Section 5.4.1, it does not omit any necessary upper bound constraint verification or incur any unnecessary verification. Hence, both $O_{css\_satr}$ and $U_{css\_satr}$ are 0.

We now conduct quantitative effectiveness analysis, i.e. analysis of omitted upper bound constraint verification, in Section 5.4.2.1. Then, we perform quantitative efficiency analysis, i.e. analysis of unnecessary upper bound constraint verification, in Section 5.4.2.2.

5.4.2.1 Quantitative effectiveness analysis

For $CSS_1$, $CSS_2$, $CSS_3$, $CSS_4$, $CSS_5$, $CSS_6$ and $CSS_7$, we first discuss their omitted temporal verification for one upper bound constraint, say $U_i$ ($i=1, 2, \ldots, N$). Then, we can sum up for all $N_{WC}$ upper bound constraints. According to Sections 2.3 and 5.4.1, we can derive the omitted upper bound constraint verification of $CSS_1$, $CSS_2$, $CSS_3$, $CSS_4$, $CSS_5$, $CSS_6$ and $CSS_7$ on $U_i$, as depicted in Table 5.1.

Table 5.1 Omitted upper bound constraint verification of $CSS_1$, $CSS_2$, $CSS_3$, $CSS_4$, $CSS_5$, $CSS_6$ and $CSS_7$ on $U_i$ ($i=1, 2, \ldots, N$)

<table>
<thead>
<tr>
<th>$CSS_i$</th>
<th>$O_{css_i}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$CSS_1$</td>
<td>$O_{css_1} = 0$</td>
</tr>
<tr>
<td>$CSS_2$</td>
<td>$O_{css_2} = 0$</td>
</tr>
<tr>
<td>$CSS_3$</td>
<td>$O_{css_3} = (1 - Q_2)\cdot [X+(i-1)\cdot Y-L]\cdot [X+(i-1)\cdot Y]$</td>
</tr>
<tr>
<td>$CSS_4$</td>
<td>$O_{css_4} = (1 - Q_2)\cdot [X+(i-1)\cdot Y-M]\cdot [X+(i-1)\cdot Y]$</td>
</tr>
</tbody>
</table>
CSS₅ \quad O_{css₅} = (Q₁ - Q₂) \cdot [X + (i-1) \cdot Y]^2

CSS₆ \quad O_{css₆} = 0

CSS₇ \quad O_{css₇} = 0

For \( N \) WC upper bound constraints \( U₁, U₂, \ldots, U₇ \), we simply sum up the omitted verification on \( U_i \) from \( i=1 \) to \( i=N \). Then, we can derive their respective total omitted upper bound constraint verification, as depicted in Table 5.2.

**Table 5.2 Total omitted upper bound constraint verification of CSS₁, CSS₂, CSS₃, CSS₄, CSS₅, CSS₆ and CSS₇**

<table>
<thead>
<tr>
<th>CSS₁</th>
<th>( O_{css₁} = 0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSS₂</td>
<td>( O_{css₂} = 0 )</td>
</tr>
<tr>
<td>CSS₃</td>
<td>( O_{css₃} = \sum_{i=1}^{N} (1 - Q₂) \cdot [X + (i-1) \cdot Y - L] \cdot [X + (i-1) \cdot Y] )</td>
</tr>
<tr>
<td>CSS₄</td>
<td>( O_{css₄} = \sum_{i=1}^{N} (1 - Q₂) \cdot [X + (i-1) \cdot Y - M] \cdot [X + (i-1) \cdot Y] )</td>
</tr>
<tr>
<td>CSS₅</td>
<td>( O_{css₅} = \sum_{i=1}^{N} (Q₁ - Q₂) \cdot [X + (i-1) \cdot Y]^2 )</td>
</tr>
<tr>
<td>CSS₆</td>
<td>( O_{css₆} = 0 )</td>
</tr>
<tr>
<td>CSS₇</td>
<td>( O_{css₇} = 0 )</td>
</tr>
</tbody>
</table>

We now take a set of specific values to see how the formulas in Table 5.2 perform. We suppose \( X = 5, Y = 3, Q₁ = 0.95, Q₂ = 0.9, Q₃ = 0.85, Q₄ = 0.8, L = 2, M = 4, \) and \( N \) can change from 0 to 20. Note that \( Q₁, Q₂, Q₃ \) and \( Q₄ \) are selected close to 1.0. This is because statistically for most cases, an activity could be completed around its mean duration. Therefore, \( Q₁, Q₂, Q₃ \) and \( Q₄ \) must be close to 1.0. The selection of other values is random and does not affect our analysis because what we want to illustrate is the trend of how the omitted upper bound constraint verification
changes by $N$.

With $N$ changing, we list corresponding omitted upper bound constraint verification in Figure 5.5.

![Graph showing change trend of omitted upper bound constraint verification of CSSMTR, CSS1, CSS2, CSS3, CSS4, CSS5, CSS6, and CSS7](image)

**Figure 5.5 Change trend of omitted upper bound constraint verification of CSSMTR, CSS1, CSS2, CSS3, CSS4, CSS5, CSS6, and CSS7**

From Figure 5.5, we can see that $O_{css1}$, $O_{css2}$, $O_{css3}$, $O_{css4}$, and $O_{css5}$ are always 0. That is to say, CSS1, CSS2, CSS6, CSS7 and CSSMTR do not omit any upper bound constraint verification. However, when $N$ increases, $O_{css3}$, $O_{css4}$ and $O_{css5}$ also increase. This means that the more WC upper bound constraints, the more omitted verification based on CSS3, CSS4, and CSS5. Therefore, CSSMTR is more effective for
upper bound constraint verification than CSS₃, CSS₄ and CSS₅.

Particularly, from Figure 5.5, we can also see that when N gets larger, \( O_{css₁}, O_{css₄} \) and \( O_{css₅} \) get much larger. Moreover, as stated in Chapters 1 and 4, in real-world grid workflow systems, grid workflows normally take a long time to complete and consequently need a large number of upper bound constraints to control and monitor their execution in terms of time [AKMK05, DBGK+03, SKJD+04]. That is to say, in real-world grid workflow systems, normally, \( N \) is a large number. Therefore, in overall terms, compared to CSS₃, CSS₄ and CSS₅, CSS₆ can significantly improve the overall temporal verification effectiveness.

5.4.2.2 Quantitative efficiency analysis

Similar to Section 5.4.2.1, for CSS₁, CSS₂, CSS₃, CSS₄, CSS₅, CSS₆ and CSS₇, we first discuss their unnecessary temporal verification for one upper bound constraint, say \( U_i \) \((i=1, 2, \ldots, N)\). Then, we sum up for all \( N \) WC upper bound constraints. According to Sections 2.3 and 5.4.1, we can derive the unnecessary upper bound constraint verification of CSS₁, CSS₂, CSS₃, CSS₄, CSS₅, CSS₆ and CSS₇ on \( U_i \), as depicted in Table 5.3.

**Table 5.3** Unnecessary upper bound constraint verification of CSS₁, CSS₂, CSS₃, CSS₄, CSS₅, CSS₆ and CSS₇ on \( U_i \) \((i=1, 2, \ldots, N)\)

<table>
<thead>
<tr>
<th>CSS</th>
<th>( U_{css_1} = Q_2*[X+(i-1)*Y]^2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSS₂</td>
<td>( U_{css_2} = 2<em>Q_2</em>[X+(i-1)*Y]^2 )</td>
</tr>
<tr>
<td>CSS₃</td>
<td>( U_{css_3} = Q_2<em>L</em>[X+(i-1)*Y] )</td>
</tr>
<tr>
<td>CSS₄</td>
<td>( U_{css_4} = Q_2<em>M</em>[X+(i-1)*Y] )</td>
</tr>
<tr>
<td>CSS₅</td>
<td>( U_{css_5} = 0 )</td>
</tr>
<tr>
<td>CSS₆</td>
<td>( U_{css_6} = (Q_2 - Q_4)*[X+(i-1)*Y]^2 )</td>
</tr>
</tbody>
</table>
### CSS₇

\[ U_{css_7} = (Q_2 - Q_3)[X + (i-1)Y]^2 \]

For \( N \) WC upper bound constraints \( U_1, U_2, \ldots, U_N \), we simply sum up the unnecessary verification on \( U_i \) from \( i=1 \) to \( i=N \). Then, we can derive their respective total unnecessary upper bound constraint verification, as depicted in Table 5.4.

**Table 5.4 Total unnecessary upper bound constraint verification of CSS₁, CSS₂, CSS₃, CSS₄, CSS₅, CSS₆ and CSS₇**

<table>
<thead>
<tr>
<th>CSS</th>
<th>( U_{css_1} )</th>
<th>( U_{css_2} )</th>
<th>( U_{css_3} )</th>
<th>( U_{css_4} )</th>
<th>( U_{css_5} )</th>
<th>( U_{css_6} )</th>
<th>( U_{css_7} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSS₁</td>
<td>( \sum_{i=1}^{N} Q_2 \cdot [X + (i-1) \cdot Y]^2 )</td>
<td>( \sum_{i=1}^{N} 2 \cdot Q_2 \cdot [X + (i-1) \cdot Y]^2 )</td>
<td>( \sum_{i=1}^{N} Q_2 \cdot L \cdot [X + (i-1) \cdot Y] )</td>
<td>( \sum_{i=1}^{N} Q_2 \cdot M \cdot [X + (i-1) \cdot Y] )</td>
<td>( 0 )</td>
<td>( \sum_{i=1}^{N} (Q_2 - Q_4) \cdot [X + (i-1) \cdot Y]^2 )</td>
<td>( \sum_{i=1}^{N} (Q_2 - Q_3) \cdot [X + (i-1) \cdot Y]^2 )</td>
</tr>
</tbody>
</table>

We now take a set of specific values to see how the formulas in Table 5.4 perform. Similar to the quantitative effectiveness analysis in Section 5.4.2.1, we also suppose \( X = 5, Y = 3, Q_1 = 0.95, Q_2 = 0.9, Q_3 = 0.85, Q_4 = 0.8, L = 2, M = 4, \) and \( N \) can change from 0 to 20. Similar to Section 5.4.2.1, \( Q_1, Q_2, Q_3 \) and \( Q_4 \) are selected close to 1.0. The selection of other values is random and does not affect our analysis because what we want to illustrate is the trend of how the unnecessary upper bound constraint verification changes by \( N \).

With \( N \) changing, we list corresponding unnecessary upper bound constraint verification in Figure 5.6.
Figure 5.6 Change trend of unnecessary upper bound constraint verification of \(CSS_{MTR}, CSS_1, CSS_2, CSS_3, CSS_4, CSS_5, CSS_6\) and \(CSS_7\)

From Figure 5.6, we can see that \(U_{css_i}\) and \(U_{css_{MTR}}\) are always 0. That is to say, \(CSS_3\) and \(CSS_{MTR}\) do not incur any unnecessary upper bound constraint verification. However, when \(N\) increases, \(U_{css_1}, U_{css_2}, U_{css_3}, U_{css_4}, U_{css_5}\) and \(U_{css_6}\) also increase. This means that the more WC upper bound constraints, the more unnecessary upper bound constraint verification based on \(CSS_1, CSS_2, CSS_3, CSS_4, CSS_6\) and \(CSS_7\). Therefore, \(CSS_{MTR}\) is more efficient for upper bound constraint verification than \(CSS_1, CSS_2, CSS_3, CSS_4, CSS_6\) and \(CSS_7\).

Especially, from Figure 5.6, we can also see that when \(N\) gets larger, \(U_{css_1}, U_{css_2}, U_{css_3}, U_{css_4}, U_{css_5}\) and \(U_{css_6}\) get much larger. Similar to the quantitative effectiveness analysis in Section 5.4.2.1, in real-world grid workflow
systems, normally, \( N \) is a large number. Therefore, in overall terms, compared to \( CSS_1 \), \( CSS_2 \), \( CSS_3 \), \( CSS_4 \), \( CSS_5 \) and \( CSS_7 \), \( CSS_{MTR} \) can significantly improve the overall temporal verification efficiency.

### 5.5 Summary

Existing representative checkpoint selection strategies often ignore some necessary checkpoints and/or select some unnecessary ones. This would omit some necessary upper bound constraint verification and/or cause some unnecessary verification. Consequently, overall temporal verification effectiveness and efficiency of upper bound constraints could be severely impacted.

To overcome the limitations of existing representative checkpoint selection strategies, in this chapter, we have developed a new checkpoint selection strategy, named \( CSS_{MTR} \) (Minimum Time Redundancy based Checkpoint Selection Strategy). It has been rigorously proved that with \( CSS_{MTR} \), checkpoints selected dynamically along grid workflow execution are not only sufficient but also necessary. Therefore, the omitted and unnecessary upper bound constraint verification which would otherwise be incurred by the existing representative strategies can be avoided.

Specifically, a new concept of minimum time redundancy has been introduced with a method, named DOMTR (Dynamic Obtaining of Minimum Time Redundancy), on how to dynamically obtain minimum time redundancy along grid workflow execution. In addition, the relationships between minimum time redundancy and upper bound constraint consistency have been investigated. Based on DOMTR and the relationships, \( CSS_{MTR} \) has been developed.

The quantitative evaluation has shown that compared to the existing representative strategies, \( CSS_{MTR} \) can improve temporal verification effectiveness and efficiency significantly.

Once checkpoints are selected, upper bound constraint verification can be
conducted. In the next chapter, we will discuss in detail how to verify a series of upper bound constraints together.
Chapter 6

Dependency-enabled Upper Bound Constraint Verification

As discussed in Section 2.4, in conventional verification work, a series of upper bound constraints are verified independently of each other without paying attention to any dependency between them. This should be simple and straightforward to understand and operate. However, we argue that a series of upper bound constraints are often dependent on each other in terms of overall temporal verification effectiveness and efficiency. As stated in Section 2.4, generally speaking, this is because later verification may make previous verification ineffective and also later verification may utilise previous verification results to save current verification computation for better efficiency. Accordingly, in this chapter, we systematically investigate the temporal dependency between upper bound constraints and its profound impact on overall temporal verification effectiveness and efficiency. Based on this, new verification methods and algorithms are developed in order to achieve better overall temporal verification effectiveness and efficiency.

Specifically, in Section 6.1, we investigate the temporal dependency between upper bound constraints. In Section 6.2, we develop some new upper bound constraint verification methods and algorithms based on temporal dependency. In Section 6.3, we conduct a quantitative evaluation which demonstrates that our new verification methods and algorithms can improve overall temporal verification effectiveness and efficiency significantly.
6.1 Temporal dependency between upper bound constraints

As indicated in Section 3.2.4, when grid workflow execution arrives at an activity, before the execution of the activity, all previous upper bound constraints would be of either SC or WC. Correspondingly, temporal dependency consists of SC temporal dependency and WC temporal dependency. The former is based on SC state of upper bound constraints while the later is based on WC state.

We first investigate SC and WC temporal dependency between two upper bound constraints in Section 6.1.1. Then in Section 6.1.2, we extend the corresponding discussion to a series of upper bound constraints.

6.1.1 SC and WC temporal dependency between two upper bound constraints

Considering two upper bound constraints $U_1$ and $U_2$ where $U_1$ is between $a_i$ and $a_j$, and $U_2$ is between $a_{i'}$ and $a_{j'}$, based on Allen’s temporal interval logic [All83, CM00], we can conclude that there are two groups of basic relationships between $U_1$ and $U_2$. We depict them in Figures 6.1 and 6.2 respectively.

![Figure 6.1 Non-nested upper bound constraints $U_1$ and $U_2$](image)
Figure 6.2 Nested upper bound constraints $U_1$ and $U_2$

In Figure 6.1, $U_1$ and $U_2$ do not have any nesting relationships while in Figure 6.2, $U_1$ is nested in $U_2$. In real-world grid workflow systems, the scenarios in Figure 6.2 are normally more common than those in Figure 6.1. This is because we normally have at least an end-to-end upper bound constraint between the first and the last activity of a grid workflow [BAV05, EPR99]. Then, all other upper bound constraints are nested in the end-to-end one. That is to say, the nesting phenomenon is common in real-world grid workflow systems. We now discuss SC and WC temporal dependency between $U_1$ and $U_2$ in Figure 6.1, and especially in Figure 6.2.

**SC temporal dependency:**

In Figure 6.1, for Scenarios 1, 2 and 4, $U_1$ and $U_2$ are relatively independent. For Scenario 3, although $U_1$ and $U_2$ have some common activities, they still have some different activities which are independent of each other. Therefore, in Figure 6.1, $U_1$ and $U_2$ can be verified independently, i.e. there is no temporal dependency issue.

In Figure 6.2, for Scenario 5, if $u(U_2) \leq u(U_1)$, then, if $U_2$ is of SC, $U_1$ must be of SC. If $U_2$ is not of SC, we need to adjust $U_2$. This inevitably affects $U_1$ because $U_1$ is included in $U_2$. Then, we have to verify $U_1$ again. That is to say, the previous temporal verification of $U_1$ is ineffective. Therefore, in Scenario 5, we must ensure $u(U_2) < u(U_1)$. Similarly, in Scenarios 6 and 7, we must ensure $u(U_1) < u(U_2)$ as well.

Now, we consider a more complicated situation for Scenario 5 which shows that
\( u(U_i) < u(U_j) \) is not sufficient yet. In Scenario 5, based on item 1 of Definition 3.4, if \( U_i \) and \( U_j \) are of SC and \( u(U_i) < u(U_j) \), we then have inequations (6-1), (6-2) and (6-3) below.

\[
\begin{align*}
  u(U_i) &< u(U_j) \quad (6-1) \\
  D(a_{i_i}, a_{j_j}) &\leq u(U_i) \quad (6-2) \\
  D(a_{i_i}, a_{j_j}) &\leq u(U_j) \quad (6-3)
\end{align*}
\]

We consider a grid workflow where \( u(U_i) = 9.5, u(U_j) = 11, D(a_{i_i}, a_{j_j}) = 8 \) and \( D(a_{i_i}, a_{j_j}) = 10 \). Then, all inequations (6-1), (6-2) and (6-3) hold. Meanwhile, we have \( D(a_{i_i}, a_{j_j}) + u(U_i) + D(a_{j_{i+1}}, a_{j_{j+1}}) = u(U_i) + D(a_{i_i}, a_{j_j}) - D(a_{i_i}, a_{j_j}) = 9.5 + 10 - 8 = 11.5 \). Obviously, 11 is less than 11.5, which means \( u(U_j) < D(a_{i_i}, a_{j_j}) + u(U_i) + D(a_{j_{i+1}}, a_{j_{j+1}}) \). In this case, at build-time stage, if \( U_j \) is of SC, we have \( D(a_{i_i}, a_{j_j}) + D(a_{i_i}, a_{j_j}) + D(a_{j_{i+1}}, a_{j_{j+1}}) \leq u(U_j) \leq D(a_{i_i}, a_{j_j}) + u(U_i) + D(a_{j_{i+1}}, a_{j_{j+1}}) \), i.e. we have inequation (6-4) below.

\[
D(a_{i_i}, a_{j_j}) < u(U_j) \quad (6-4)
\]

According to item 1 of Definition 3.4, inequation (6-4) means that \( U_i \) is of SC. Similar to the situation where \( u(U_j) \leq u(U_i) \), if \( U_j \) is not of SC, adjusting \( U_j \) will inevitably affect the setting of \( U_i \) and we will need to verify \( U_i \) again. Then, the previous temporal verification of \( U_i \) will become ineffective. Hence, in Scenario 5, \( u(U_i) < u(U_j) \) is not sufficient and we still need to ensure \( D(a_{i_i}, a_{j_j}) + u(U_i) + D(a_{j_{i+1}}, a_{j_{j+1}}) \leq u(U_j) \). Similarly, in Scenario 6, we must ensure \( u(U_i) + D(a_{j_{i+1}}, a_{j_{j+1}}) \leq u(U_j) \), and in Scenario 7, we must ensure \( D(a_{i_i}, a_{j_j}) + u(U_i) \leq u(U_j) \).

In summary, temporal dependency based on the SC state, i.e. SC temporal
dependency, between two upper bound constraints in Scenarios 5, 6 and 7 of Figure 6.2 must be taken into consideration and consequently we have Definition 6.1 below.

**Definition 6.1**
Let $U_1$ and $U_2$ be two upper bound constraints (see Figure 6.2) where $U_1$ is between $a_{i_1}$ and $a_{j_1}$, and $U_2$ is between $a_{i_2}$ and $a_{j_2}$ ($i_2 \leq i_1 < j_1 \leq j_2$), i.e. $U_1$ is nested in $U_2$. Then, SC temporal dependency between $U_1$ and $U_2$ is said be consistent if $D(a_{i_1}, a_{i_2}) + u(U_1) + D(a_{j_1}, a_{j_2}) \leq u(U_2)$.

**WC temporal dependency:**

Similar to the discussion for SC temporal dependency, temporal dependency based on WC state, i.e. WC temporal dependency, between two upper bound constraints in Scenarios 5, 6 and 7 of Figure 6.2 must also be taken into consideration and consequently we have Definition 6.2 below.

**Definition 6.2**
Let $U_1$ and $U_2$ be two upper bound constraints (see Figure 6.2) where $U_1$ is between $a_{i_1}$ and $a_{j_1}$, and $U_2$ is between $a_{i_2}$ and $a_{j_2}$ ($i_2 \leq i_1 < j_1 \leq j_2$), i.e. $U_1$ is nested in $U_2$. Then, WC temporal dependency between $U_1$ and $U_2$ is said be consistent if $M(a_{i_1}, a_{i_2}) + u(U_1) + M(a_{j_1}, a_{j_2}) \leq u(U_2)$.

Definitions 6.1 and 6.2 cover Scenarios 5, 6 and 7 of Figure 6.2. If $i_2< i_1$ and $j_1< j_2$, it corresponds to Scenario 5. If $i_2= i_1$, it corresponds to Scenario 6. If $j_1= j_2$, it corresponds to Scenario 7.

**6.1.2 SC and WC temporal dependency between a series of upper bound constraints**

We now further investigate SC and WC temporal dependency between a series of upper bound constraints. Based on the discussion in Section 6.1.2 for two upper bound constraints, we can see that if there are no nesting relationships, there will be
no temporal dependency issue involved. Therefore, we only need to consider the situation where a series of upper bound constraints have nesting relationships. Considering $N$ upper bound constraints $U_1$, $U_2$, ..., $U_N$, based on the nesting relationships between two upper bound constraints in Scenarios 5, 6 and 7 of Figure 6.2, we can derive four general groups of basic nesting relationships. We depict them in Figure 6.3. Other relationships can be composed from them or Scenarios 1-7, so their corresponding discussion of temporal dependency would be similar.

In Figure 6.3, Scenarios 8, 9 and 10 are actually the extension of Scenarios 5, 6 and 7 of Figure 6.2 respectively. Scenario 11 is a combination of Scenarios 8, 9 and 10.

To discuss SC and WC temporal dependency in Scenarios 8, 9, 10 and 11, we derive Theorem 6.1 below. Based on Theorem 1, SC or WC temporal dependency in
Scenarios 8, 9, 10 and 11 can be translated into that in Scenarios 5, 6 and 7 of Figure 6.2 respectively.

**Theorem 6.1**

Let $U_1, U_2, \ldots, U_N$ be $N$ upper bound constraints (see Figure 6.3) where $U_1$ is between $a_i$ and $a_j$, $U_2$ is between $a_i$ and $a_j$, and so forth ($i_1 \leq \ldots \leq i_2 \leq \ldots \leq j_1 \leq \ldots \leq j_2 \leq \ldots \leq j_N$), i.e. $U_k$ is nested in $U_{k+1}$ ($1 \leq k \leq N-1$). Then,

1) if SC temporal dependency between any two adjacent upper bound constraints $U_k$ and $U_{k+1}$ is consistent, SC temporal dependency between any two non-adjacent upper bound constraints must also be consistent;

2) if WC temporal dependency between any two adjacent upper bound constraints $U_k$ and $U_{k+1}$ is consistent, WC temporal dependency between any two non-adjacent upper bound constraints must also be consistent.

**Proof:**

1) We conduct the proof for Scenario 8 here. The corresponding proof for Scenarios 9, 10 and 11 is similar. For simplicity, we consider $U_1$, $U_2$ and $U_3$. Suppose temporal dependency between $U_1$ and $U_2$ is consistent and temporal dependency between $U_2$ and $U_3$ is consistent. Now we prove that temporal dependency between $U_1$ and $U_3$ is also consistent. That is to say, the consistency of temporal dependency is transitive. According to Definition 6.1, we have inequations (6-5) and (6-6) below.

\[
D(a_{i_2}, a_{i_2-1}) + u(U_1) + D(a_{j_1}, a_{j_1}) \leq u(U_2) \quad (6-5)
\]

\[
D(a_{i_1}, a_{i_1-1}) + u(U_2) + D(a_{j_1}, a_{j_1}) \leq u(U_3) \quad (6-6)
\]

Based on inequations (6-5) and (6-6), we have:
\[
D(a_{i_1}, a_{i_1-1}) + u(U_1) + D(a_{j_1}, a_{j_1}) + D(a_{j_1}, a_{i_1}) + u(U_1) + D(a_{j_1}, a_{j_1}) + D(a_{j_1}, a_{j_1}) \leq D(a_{i_1}, a_{i_1-1}) + u(U_2) + D(a_{j_1}, a_{j_1}) \leq u(U_3). \]

Then, we have inequation (6-7) below.

82
According to Definition 6.1, inequation (6-7) means that temporal dependency between $U_1$ and $U_3$ is consistent.

2) The proof is similar to 1), hence omitted.

Thus, in overall terms, the theorem holds. □

### 6.2 Upper bound constraint verification based on temporal dependency

In this section, we discuss how to conduct temporal verification of upper bound constraints based on their SC and WC temporal dependency for better overall verification effectiveness and efficiency.

At build-time stage, according to Section 6.1, for better temporal verification effectiveness, besides upper bound constraints themselves, we still need to verify SC and WC temporal dependency between them. Regarding temporal verification efficiency, since the consistency of SC and WC temporal dependency is still unknown for the moment, we cannot utilise it to use the previous temporal verification results to reduce the later temporal verification computation. That is to say, temporal verification efficiency cannot be improved. We may note that verifying temporal dependency would cause some extra computation. However, the extra computation is negligible. For example, we consider Scenario 5 of Figure 6.2 for SC temporal dependency. To verify the temporal dependency between $U_1$ and $U_2$, according to Definition 6.1, we need to conduct the inequation $D(a_{i_1}, a_{i_{-1}}) + u(U_1) + D(a_{j_{+1}}, a_{j_{-1}}) \leq u(U_2)$. Apparently, $u(U_1)$ and $u(U_2)$ are given. Now, we consider $D(a_{i_1}, a_{i_{-1}})$ and $D(a_{j_{+1}}, a_{j_{-1}})$. Since $U_2$ is definitely verified no matter whether we verify the temporal dependency between $U_1$ and $U_2$, according to item 1
of Definition 3.4, \( D(a_{i_1}, a_{j_1}) \) must be computed. From Scenario 5 of Figure 6.2, we can see that \( D(a_{i_1}, a_{i-1}) \) and \( D(a_{j_{i+1}}, a_{j_2}) \) are part of \( D(a_{i_1}, a_{j_2}) \). That is to say, \( D(a_{i_1}, a_{i-1}) \) and \( D(a_{j_{i+1}}, a_{j_2}) \) can be derived from the verification of \( U_2 \). Therefore, the extra computation caused by verifying temporal dependency would be one or two additions and/or comparisons. However, in Section 6.2.1, we will see that we need to conduct a large amount of verification computation for verifying many upper bound constraints at various activities. Compared to this, the one or two additions and/or comparisons would be negligible. Therefore, temporal verification efficiency is about the same as that in the conventional verification work.

At run-time instantiation stage, grid workflow instances are enacted and we get the absolute start time, i.e. \( S(a_i) \). However, according to Definitions 3.5, 6.1 and 6.2, the verification of upper bound constraints and their SC and WC temporal dependency has nothing to do with \( S(a_i) \). Therefore, we need not verify them again and simply take the build-time verification results. Correspondingly, the build-time improvement on temporal verification effectiveness can be kept while temporal verification efficiency is about the same as that in the conventional verification work.

At run-time execution stage, since at build-time stage we have already verified SC and WC temporal dependency and consequently we can remove any of their inconsistency, the ineffectiveness case of build-time stage will not occur again. That is to say, SC and WC temporal dependency will not improve temporal verification effectiveness further, but will keep the build-time improvement. However, we will see later in Section 6.2.2 that based on SC and WC temporal dependency, we can utilise the previous verification results to reduce later verification computation. That is to say, we are able to conduct temporal verification more efficiently than that without considering SC and WC temporal dependency.

Now we detail the temporal verification process based on SC and WC temporal dependency at build-time stage in Section 6.2.1 and that at run-time execution stage in Section 6.2.2.
6.2.1 Upper bound constraint verification at build-time stage

On one hand, for each upper bound constraint, say $U(a_i, a_j)$, we conduct its verification according to Definition 3.4. We compute $D(a_i, a_j)$, $M(a_i, a_j)$ and $d(a_i, a_j)$. Then, we conduct corresponding comparisons according to items 1, 2, 3 and 4 of Definition 3.4 to judge whether $U(a_i, a_j)$ is of SC, WC, WI or SI.

On the other hand, we verify SC and WC temporal dependency between upper bound constraints according to Section 6.1.

- For Scenarios 1, 2, 3 and 4 of Figure 6.1, as indicated in Section 6.1, there is no temporal dependency issue.

- In Figure 6.2, for Scenario 5, according to Definition 6.1, if $D(a_{i_1}, a_{i_2}) + D(a_{j_1}, a_{j_2}) + u(U_1) \leq u(U_2)$, SC temporal dependency between $U_1$ and $U_2$ is consistent. Otherwise, it is inconsistent. For WC temporal dependency, according to Definition 6.2, if $M(a_{i_1}, a_{i_2}) + M(a_{j_1}, a_{j_2}) + u(U_1) \leq u(U_2)$, WC temporal dependency between $U_1$ and $U_2$ is consistent. Otherwise, it is inconsistent. $D(a_{i_1}, a_{i_2})$, $D(a_{j_1}, a_{j_2})$, $M(a_{i_1}, a_{i_2})$ and $M(a_{j_1}, a_{j_2})$ can be derived during the verification of $U_1$ and $U_2$. For Scenarios 6 and 7, the corresponding verification of SC and WC temporal dependency is similar to that for Scenario 5 as they can be viewed as special cases of Scenario 5.

- For scenarios of Figure 6.3, according to Theorem 6.1, we only need to verify SC and WC temporal dependency between any two adjacent upper bound constraints. Hence, the corresponding verification is also similar to that for Scenario 5 of Figure 6.2.

In overall terms, for each scenario of Figures 6.1, 6.2 and 6.3, we can derive a specific verification algorithm. However, we can see that Scenario 8 is more representative than other scenarios. Therefore, we only give the verification algorithm for Scenario 8. We depict it in Algorithm 6.1. Algorithm 6.1 can be
rendered for other scenarios.

| **Input** | ArrayEU: an array of upper bound constraints whose relationships are like the situation in Scenario 8 of Figure 6.3; Maximum durations of all activities; |
| **Output** | SC, WC, WI and SI report for upper bound constraints; Consistent or inconsistent report for SC and WC temporal dependency between upper bound constraints; |
| **Step 1** | Consider current two adjacent upper bound constraints from ArrayEU. Initially, they are the first two items of ArrayEU. Select current two adjacent upper bound constraints from ArrayEU to $U_k$ and $U_{k+1}$ respectively where $U_k$ is nested in $U_{k+1}$, $U_k$ is between $a_{i_k}$ and $a_{j_k}$ and $U_{k+1}$ is between $a_{i_{k+1}}$ and $a_{j_{k+1}}$ $(i_k < j_k < j_{k+1})$. |
| **Step 2** | Verify $U_k$ |
| **Step 2** | If $(D(a_{i_k}, a_{j_k}) \leq u(U_k))$ then Output “$U_k$ is of SC” Else if $(M(a_{i_k}, a_{j_k}) \leq u(U_k) < D(a_{i_k}, a_{j_k}))$ then Output “$U_k$ is of WC” Else if $(d(a_{i_k}, a_{j_k}) \leq u(U_k) < M(a_{i_k}, a_{j_k}))$ then Output “$U_k$ is of WI” Else // $u(U_k) < d(a_{i_k}, a_{j_k})$ Output “$U_k$ is of SI” End if |

86
Step 3  Verify $U_{k+1}$

If ($D(a_{i_{k+1}}, a_{j_{k+1}}) \leq u(U_{k+1})$) then

Output “$U_{k+1}$ is of SC”

Else if ($M(a_{i_{k+1}}, a_{j_{k+1}}) \leq u(U_{k+1}) < D(a_{i_{k+1}}, a_{j_{k+1}})$) then

Output “$U_{k+1}$ is of WC”

Else if ($d(a_{i_{k+1}}, a_{j_{k+1}}) \leq u(U_{k+1}) < M(a_{i_{k+1}}, a_{j_{k+1}})$) then

Output “$U_{k+1}$ is of WI”

Else // $u(U_{k+1}) < d(a_{i_{k+1}}, a_{j_{k+1}})$

Output “$U_{k+1}$ is of SI”

End if

Step 4  Verify SC and WC temporal dependency between $U_k$ and $U_{k+1}$

If ($D(a_{i_{k+1}}, a_{j_{k+1}}) + D(a_{j_{k+1}}, a_{j_{k+1}}) + u(U_k) \leq u(U_{k+1})$) then

Output “SC temporal dependency between $U_k$ and $U_{k+1}$ is consistent”

Else

Output “SC temporal dependency between $U_k$ and $U_{k+1}$ is inconsistent”

End if

If ($M(a_{i_{k+1}}, a_{j_{k+1}}) + M(a_{j_{k+1}}, a_{j_{k+1}}) + u(U_k) \leq u(U_{k+1})$) then

Output “WC temporal dependency between $U_k$ and $U_{k+1}$ is consistent”

Else

Output “WC temporal dependency between $U_k$ and $U_{k+1}$ is inconsistent”

End if
Algorithm 6.1 Build-time upper bound constraint verification based on SC and WC temporal dependency for Scenario 8 of Figure 6.3

6.2.2 Upper bound constraint verification at run-time execution stage

According to Sections 3.2.1 and 3.2.3, at a checkpoint, if an upper bound constraint does not cover the checkpoint, then the grid workflow execution at the checkpoint does not affect its consistency. Therefore, when we conduct temporal verification of upper bound constraints at a checkpoint, we only need to consider those upper bound constraints which cover it.

To verify current consistency of upper bound constraint $U(a_i, a_j)$ at checkpoint $a_p$ between $a_i$ and $a_j$, we firstly check the execution of each activity. If each activity from $a_i$ to $a_p$ is completed within its maximum duration, according to the build-time verification, previous SC $U(a_i, a_j)$ is still of SC after execution of $a_p$. For all other situations, we conduct verification of $U(a_i, a_j)$ according to Definition 3.6 to judge whether $U(a_i, a_j)$ is of SC, WC, WI or SI after execution of $a_p$.

For those upper bound constraints which have no nesting relationships, namely like the scenarios in Figure 6.1, we conduct the above verification process of $U(a_i, a_j)$ for each of them. For those ones which have nesting relationships, firstly for Scenario 5 of Figure 6.2, we derive Theorem 6.2 below.

Theorem 6.2
Let $U_1$ and $U_2$ be two upper bound constraints (see Scenario 5 of Figure 6.2) where $U_1$ is between $a_i$ and $a_j$, and $U_2$ is between $a_h$ and $a_j$, ($i_2 < i_1 < j_1 < j_2$), namely $U_1$
is nested in $U_2$. Then, at checkpoint $a_p$ between $a_i$ and $a_j$,

1) if $R(a_i, a_{i-1}) \leq D(a_i, a_{i-1})$ and $U_1$ is of SC, $U_2$ must also be of SC;

2) if $R(a_i, a_{i-1}) \leq M(a_i, a_{i-1})$ and $U_1$ is of WC, $U_2$ must also be of WC or even SC.

**Proof:**

1) If $U_1$ is of SC and SC temporal dependency between $U_1$ and $U_2$ is consistent, we have inequations (6-8) and (6-9) below.

$$R(a_i, a_p) + D(a_{p+1}, a_j) \leq u(U_i) \tag{6-8}$$

$$D(a_i, a_{i-1}) + u(U_i) + D(a_{i+1}, a_j) \leq u(U_2) \tag{6-9}$$

If $R(a_i, a_{i-1}) \leq D(a_i, a_{i-1})$, based on inequations (6-8) and (6-9), we have $u(U_2) \geq D(a_i, a_{i-1}) + u(U_i) + D(a_{i+1}, a_j) \geq R(a_i, a_{i-1}) + u(U_i) + D(a_{i+1}, a_j) = R(a_i, a_p) + D(a_{p+1}, a_j)$. Finally, we have inequation (6-10) below.

$$R(a_i, a_p) + D(a_{p+1}, a_j) \leq u(U_2) \tag{6-10}$$

According to item 1 of Definition 3.6, inequation (6-10) means that $U_2$ is of SC.

2) The proof is similar to 1), hence omitted.

Thus, in overall terms, the theorem holds.

Based on Theorem 6.2, when we verify $U_1$ and $U_2$ in Scenario 5 of Figure 6.2, we first verify $U_1$ according to Definition 3.6. If $U_1$ is of WI or SI, we also verify $U_2$ according to Definition 3.6. If $U_1$ is of SC, we compare $R(a_i, a_{i-1})$ with $D(a_i, a_{i-1})$. Then, if $R(a_i, a_{i-1}) \leq D(a_i, a_{i-1})$, then $U_2$ is of SC. If $U_1$ is of WC, we
compare $R(a_{i_z}, a_{i_{z-1}})$ with $M(a_{i_z}, a_{i_{z-1}})$. If $R(a_{i_z}, a_{i_{z-1}}) \leq M(a_{i_z}, a_{i_{z-1}})$, $U_2$ is of WC or SC. For all other situations, we verify $U_2$ according to Definition 3.6. We need not compute $M(a_{i_z}, a_{i_{z-1}})$ and $D(a_{i_z}, a_{i_{z-1}})$ because we can get their values directly from the build-time upper bound constraint verification.

At the same time, in theory, we can also verify $U_1$ and $U_2$ only according to Definition 3.6, rather than based on Definition 3.6 and Theorem 6.2 together. However, in practice, the temporal verification based on Definition 3.6 and Theorem 6.2 together is more efficient than that based on Definition 3.6 alone. The reasons are as follows. Firstly, for $U_1$, these two approaches are the same. Secondly, if $U_1$ is of WI or SI, or $U_1$ is of SC and $R(a_{i_z}, a_{i_{z-1}}) > D(a_{i_z}, a_{i_{z-1}})$ or $U_1$ is of WC and $R(a_{i_z}, a_{i_{z-1}}) > M(a_{i_z}, a_{i_{z-1}})$, these two approaches are still the same. Thirdly, if $U_1$ is of SC and $R(a_{i_z}, a_{i_{z-1}}) \leq D(a_{i_z}, a_{i_{z-1}})$, based on Theorem 6.2, $U_2$ must be of SC. If $U_1$ is of WC and $R(a_{i_z}, a_{i_{z-1}}) \leq M(a_{i_z}, a_{i_{z-1}})$, based on Theorem 6.2, $U_2$ must be of WC or SC. According to Section 3.2.4, WC can be carried forward. Therefore, we need not conduct any further verification computation. However, if we verify $U_2$ directly based on Definition 3.6, we need to compute $R(a_{i_z}, a_{i_{z-1}})$, $R(a_{i_z}, a_{p})$, $D(a_{p+1}, a_{j_z})$, $D(a_{j_z+1}, a_{j_z})$, $M(a_{p+1}, a_{j_z})$ and $M(a_{j_z+1}, a_{j_z})$. Then, we conduct corresponding comparisons with $u(U_2)$ according to Definition 3.6 to judge whether $U_2$ is of SC, WC, WI or SI. Clearly, if based on Definition 3.6 alone, we need to conduct more computation. That is to say, the temporal verification based on Definition 3.6 and Theorem 6.2 together is more efficient than that based on Definition 3.6 alone.

For Scenarios 6 and 7 of Figure 6.2, we can view them as two special cases of Scenario 5. Hence, we can equally apply Theorem 6.2 to them so that we can also conduct the corresponding temporal verification more efficiently. Especially, for Scenario 6, because $U_1$ and $U_2$ have the same start point, we can derive Corollary 6.1 below. With Corollary 6.1, temporal verification efficiency can be further improved.
Corollary 6.1

Let $U_1$ and $U_2$ be two upper bound constraints (see Scenario 6 of Figure 6.2) where $U_1$ is between $a_i$ and $a_j$ and $U_2$ is between $a_i$ and $a_j$ ($i < j_1 < j_2$), namely $U_1$ and $U_2$ have the same start point $a_i$ and $U_1$ is nested in $U_2$. Then, at checkpoint $a_p$ between $a_i$ and $a_j$,

1) if $U_1$ is of SC, $U_2$ must also be of SC;
2) if $U_1$ is of WC, $U_2$ must also be of WC or even SC.

Proof:

1) Since $U_1$ and $U_2$ have the same start point $a_i$, we have $R(a_i, a_{i-1}) = D(a_i, a_{i-1}) = 0$. Hence, we always have $R(a_i, a_{i-1}) \leq D(a_i, a_{i-1})$.

2) The proof is similar to 1), hence omitted.

Thus, according to Theorem 6.2, the corollary holds.

Based on Corollary 6.1, when we verify $U_1$ and $U_2$ in Scenario 6 of Figure 6.2, we verify $U_1$ first. If $U_1$ is of SC, we can draw an immediate conclusion that $U_2$ is also of SC. And if $U_1$ is of WC, $U_2$ must also be of WC or even SC. Consequently, the verification process based on Corollary 6.1 is simpler than that based on Theorem 6.2 as we need not compare $R(a_i, a_{i-1})$ and $D(a_i, a_{i-1})$. So, with Corollary 6.1, the verification efficiency can be further improved.

Since Theorem 6.2 is based on the temporal dependency between upper bound constraints and Corollary 6.1 is derived from Theorem 6.2, Corollary 6.1 is also based on the temporal dependency between upper bound constraints. Hence, we can say that based on the temporal dependency between upper bound constraints, we can conduct temporal verification of upper bound constraints in Figure 6.2 more efficiently.
In Figure 6.3, for Scenarios 8, 9, 10 and 11, between any two upper bound constraints, we can also apply Theorem 6.2 to them so that we can improve the temporal verification efficiency. Especially, for Scenario 9, since all upper bound constraints have the same start point, based on Corollary 6.1, we can derive Corollary 6.2 below. With Corollary 6.2, the temporal verification efficiency can be further improved.

**Corollary 6.2**

Let \( U_1, U_2, \ldots, U_N \) be \( N \) upper bound constraints (see Scenario 9 of Figure 6.3) where \( U_1 \) is between \( a_i \) and \( a_{i+1} \), \( U_2 \) is between \( a_i \) and \( a_{i+2} \) and so forth \((i_1 < j_1 < j_2 < \ldots < j_N)\), namely \( U_1, U_2, \ldots, U_N \) have the same start point \( a_i \) and \( U_i \) is nested in \( U_{i+1} \), \( U_{i+1} \) is nested in \( U_{i+2} \) and so forth. Then, at checkpoint \( a_p \) between \( a_i \) and \( a_{j_N} \),

1) if \( U_k \) is of SC, any upper bound constraint \( U_s \) after \( U_k \) must also be of SC \((k < s \leq N)\);

2) if \( U_k \) is of WC, any upper bound constraint \( U_s \) after \( U_k \) must also be of WC or even SC \((k < s \leq N)\).

**Proof:**

1) Considering \( U_s \) and \( U_k \) together, according to Corollary 6.1, if \( U_k \) is of SC, we do have \( U_s \) as SC.

2) The proof is similar to 1), hence omitted.

Thus, the corollary holds.

Based on Corollary 6.2, when we verify upper bound constraints in Scenario 9 of Figure 6.3, we verify them one after another until we meet an SC or WC one or finish all of them. Once we meet an SC or WC upper bound constraint, we need not conduct further verification for any other upper bound constraints after it. In contrast, if we verify upper bound constraints in Scenario 9 of Figure 3 based on Definition 3.6, we have to verify all of them. And if based on Theorem 6.2, we have to verify each pair of upper bound constraints. Apparently, the temporal verification based on Corollary 6.2 is more efficient than that based on Definition 3.6 or Theorem 6.2.
Since Theorem 6.2 is based on the temporal dependency between upper bound constraints and Corollary 6.2 is derived from Corollary 6.1 which is derived from Theorem 6.2, Corollary 6.2 is also based on the temporal dependency. Therefore, we can say that based on the temporal dependency between upper bound constraints, we can conduct temporal verification of upper bound constraints in Figure 6.3 more efficiently.

In overall terms, for each scenario of Figures 6.1, 6.2 and 6.3, we can derive a specific verification algorithm. Similar to the build-time upper bound constraint verification, we only depict the representative one for Scenario 8 of Figure 6.3 in Algorithm 6.2. Algorithm 6.2 can be rendered for other scenarios.

<table>
<thead>
<tr>
<th>Input</th>
<th>ArrayEB: an array of previous SC and WC upper bound constraints which cover checkpoint $a_p$ and whose temporal dependency is like the situation in Scenario 8 of Figure 6.3; Maximum durations of all activities involved in upper bound constraints in ArrayEB;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>SC, WC, WI or SI report for upper bound constraints</td>
</tr>
<tr>
<td>Step 1</td>
<td>Consider current two adjacent upper bound constraints from ArrayEB. Initially, they are the first two items of ArrayEB. Select current two adjacent upper bound constraints from ArrayEB to $U_k$ and $U_{k+1}$ respectively where $U_k$ is nested in $U_{k+1}$, $U_k$ is between $a_{i_k}$ and $a_{j_k}$ and $U_{k+1}$ is between $a_{i_{k+1}}$ and $a_{j_{k+1}}$ ($i_k &lt; i &lt; p &lt; j &lt; j_{k+1}$).</td>
</tr>
<tr>
<td>Step 2</td>
<td>Verify $U_k$</td>
</tr>
</tbody>
</table>
If \((R(a_n) \leq D(a_n))\) for all \(a_n\) between \(a_i\) and \(a_p, i \leq n \leq p\) then

If \((U_k\) is previously of SC) 

Output “\(U_k\) is still of SC after execution of \(a_p\)”

Else // \(U_k\) is previously of WC

If \(( R(a_{i_k}, a_p) + D(a_{p+1}, a_{j_k}) \leq u(U_k) )\) then

Output “\(U_k\) is of SC”

Else if \(( R(a_{i_k}, a_p) + M(a_{p+1}, a_{j_k}) \leq u(U_k) < R(a_{i_k}, a_p) + D(a_{p+1}, a_{j_k}) )\) then

Output “\(U_k\) is of WC”

Else if \(( R(a_{i_k}, a_p) + d(a_{p+1}, a_{j_k}) \leq u(U_k) < R(a_{i_k}, a_p) + M(a_{p+1}, a_{j_k}) )\) then

Output “\(U_k\) is of WI”

Else // \(( R(a_{i_k}, a_p) + d(a_{p+1}, a_{j_k}) < u(U_k) )\)

Output “\(U_k\) is of SI”

End if

End if

Else if \(( R(a_{i_k}, a_p) + D(a_{p+1}, a_{j_k}) \leq u(U_k) )\) then

Output “\(U_k\) is of SC”

Else if \(( R(a_{i_k}, a_p) + M(a_{p+1}, a_{j_k}) \leq u(U_k) < R(a_{i_k}, a_p) + D(a_{p+1}, a_{j_k}) )\) then

Output “\(U_k\) is of WC”

Else if \(( R(a_{i_k}, a_p) + d(a_{p+1}, a_{j_k}) \leq u(U_k) < R(a_{i_k}, a_p) + M(a_{p+1}, a_{j_k}) )\) then
Output “$U_k$ is of WI”

Else // ( $R(a_{i_k}, a_p) + d(a_{p+1}, a_{j_k}) < u(U_k)$)

Output “$U_k$ is of SI”

End if

<table>
<thead>
<tr>
<th>Step 3</th>
<th>Verify $U_{k+j}$</th>
</tr>
</thead>
</table>

If ($U_k$ is of WI or SI) then

If ( $R(a_{i_k}, a_p) + D(a_{p+1}, a_{j_k}) \leq u(U_{k+j})$) then,

Output “$U_{k+j}$ is of SC”

Else if ( $R(a_{i_k}, a_p) + M(a_{p+1}, a_{j_k}) \leq u(U_{k+j}) < R(a_{i_k}, a_p) + D(a_{p+1}, a_{j_k})$)

Output “$U_{k+j}$ is of WC”

Else if ( $R(a_{i_k}, a_p) + d(a_{p+1}, a_{j_k}) \leq u(U_{k+j}) < R(a_{i_k}, a_p) + M(a_{p+1}, a_{j_k})$)

Output “$U_{k+j}$ is of WI”

Else // ($u(U_{k+j}) < R(a_{i_k}, a_p) + d(a_{p+1}, a_{j_k})$)

Output “$U_{k+j}$ is of SI”

End if

Else if ($U_k$ is of SC)

If ( $R(a_{i_k}, a_{j_k}) \leq D(a_{i_k}, a_{j_k-1})$) then

Output “$U_{k+j}$ is of SC”

Else if ( $R(a_{i_k}, a_p) + D(a_{p+1}, a_{j_k}) \leq u(U_{k+j})$) then,

Output “$U_{k+j}$ is of SC”

Else if ( $R(a_{i_k}, a_p) + M(a_{p+1}, a_{j_k}) \leq u(U_{k+j}) <$
\[ R(a_{i_{k+1}}, a_p) + D(a_{p+1}, a_{j_{k+1}}) \]

Output “\( U_{k+j} \) is of WC”

Else if ( \( R(a_{i_{k+1}}, a_p) + d(a_{p+1}, a_{j_{k+1}}) \leq u(U_{k+j}) < R(a_{i_{k+1}}, a_p) + M(a_{p+1}, a_{j_{k+1}}) \))

Output “\( U_{k+j} \) is of WI”

Else // (\( u(U_{k+j}) < R(a_{i_{k+1}}, a_p) + d(a_{p+1}, a_{j_{k+1}}) \))

Output “\( U_{k+j} \) is of SI”

End if

Else // (\( U_k \) is of WC)

If ( \( R(a_{i_{k+1}}, a_{i_{k-1}}) \leq M(a_{i_{k+1}}, a_{i_{k-1}}) \)) then

Output “\( U_{k+j} \) is of WC or even SC”

Else if ( \( R(a_{i_{k+1}}, a_p) + D(a_{p+1}, a_{j_{k+1}}) \leq u(U_{k+j}) \)) then,

Output “\( U_{k+j} \) is of SC”

Else if ( \( R(a_{i_{k+1}}, a_p) + M(a_{p+1}, a_{j_{k+1}}) \leq u(U_{k+j}) < R(a_{i_{k+1}}, a_p) + D(a_{p+1}, a_{j_{k+1}}) \))

Output “\( U_{k+j} \) is of WC”

Else if ( \( R(a_{i_{k+1}}, a_p) + d(a_{p+1}, a_{j_{k+1}}) \leq u(U_{k+j}) < R(a_{i_{k+1}}, a_p) + M(a_{p+1}, a_{j_{k+1}}) \))

Output “\( U_{k+j} \) is of WI”

Else // (\( u(U_{k+j}) < R(a_{i_{k+1}}, a_p) + d(a_{p+1}, a_{j_{k+1}}) \))

Output “\( U_{k+j} \) is of SI”

End if

End if
Step 4

Go back to Step 1 until the completion of all upper bound constraints in ArrayEB

Algorithm 6.2 Upper bound constraint verification based on temporal dependency at checkpoint $a_p$ at run-time execution stage for Scenario 8 of Figure 6.3

6.3 Comparison and quantitative evaluation

Compared with the existing conventional temporal verification work, the clear difference in our research is that we have systematically investigated temporal dependency between upper bound constraints and have applied it to temporal verification of upper bound constraints. By doing so, according to Section 6.2 we can achieve better temporal verification effectiveness at build-time stage while keeping temporal verification efficiency about the same as that in the conventional verification work. At run-time instantiation stage, the situation is similar as we need not conduct any further temporal verification and simply take build-time verification results. At run-time execution stage, we can achieve better verification efficiency while keeping the build-time improvement on temporal verification effectiveness. Hence, in overall terms, based on temporal dependency we can achieve better overall temporal verification effectiveness and efficiency.

We now conduct further quantitative analysis so that we can get a specific picture of how temporal dependency impacts temporal verification effectiveness at build-time stage and temporal verification efficiency at run-time execution stage. We take Scenario 8 of Figure 6.3 as an example because in Figure 6.1, Figure 6.2 and Figure 6.3, we can see that it is more representative than other scenarios. According to Definition 3.6, the primary upper bound constraint verification computation is focused on the sum of maximum durations between two activities. Therefore, we take each maximum duration addition operation as a verification computation unit. Correspondingly, we conduct the analysis of temporal verification effectiveness and efficiency in terms of the number of such units.
We conduct quantitative effectiveness analysis in Section 6.3.1 and quantitative efficiency analysis in Section 6.3.2.

### 6.3.1 Quantitative effectiveness analysis

At build-time stage, in Scenario 8 of Figure 6.3, we denote the number of ineffective verification computation units based on temporal dependency and that without based on temporal dependency as follows.

- \( \text{effe}_{\text{with}} \) : number of ineffective verification computation units based on temporal dependency.

- \( \text{effe}_{\text{without}} \) : number of ineffective verification computation units without based on temporal dependency.

Then, our improvement on temporal verification effectiveness based on temporal dependency is reflected by comparing \( \text{effe}_{\text{with}} \) with \( \text{effe}_{\text{without}} \).

According to Section 6.1, if we ignore temporal dependency, there may be some inconsistent temporal dependency. We denote the number of inconsistent temporal dependency as \( L \), i.e. we have:

- \( L \) : number of inconsistent temporal dependency.

For each inconsistent temporal dependency between two upper bound constraints, we need to adjust and re-verify one or both of them. Hence, the previous temporal verification is ineffective. For simplicity, we assume that we only need to adjust and re-verify the inner one. Then, we will have \( L \) upper bound constraints whose temporal verification is ineffective. Among such \( L \) upper bound constraints, we make the following denotations.
• $P_1$: number of activities covered by the first upper bound constraint of $L$.

• $Q_1$: number of activities between two adjacent upper bound constraints of $L$.

For simplicity, we assume that any two adjacent upper bound constraints of $L$ have the same $Q_1$. Then, for the $r^{th}$ upper bound constraint of $L$ ($r \leq L$), the number of activities between its start point and the start point of the $1^{st}$ upper bound constraint is $(r-1)Q_1$ and the number of activities between its end point and the end point of the $1^{st}$ upper bound constraint is also $(r-1)Q_1$. Hence, for the $r^{th}$ upper bound constraint, we can derive that the number of ineffective verification computation units is $P_1 + 2(r-1)Q_1$. Then, we have formula (6-11) below.

$$effe_{without} = \sum_{r=1}^{L} [P_1 + 2(r-1)Q_1] = L \cdot P_1 + L \cdot (L-1) \cdot Q_1 \quad (6-11)$$

Regarding $effe_{with}$, according to Section 6.1, $L$ is 0 as inconsistency of temporal dependency can be removed by taking temporal dependency into verification. Therefore, the above ineffective temporal verification of $effe_{without}$ will not incur. Hence, $effe_{with} = 0$.

We now take a set of values to quantitatively illustrate $effe_{with}$ and $effe_{without}$. Suppose $Q_1 = 2$, $P_1 = 5$, and the number of all temporal dependency between upper bound constraints is 20. The selection of these values is rather random and does not affect our illustration because what we want to illustrate is the trend of how $effe_{with}$ is less than $effe_{without}$ when $L$ changes. With $L$ changing, we depict corresponding $effe_{with}$ and $effe_{without}$ in Figure 6.4.
Figure 6.4 Change trend of ineffective upper bound constraint verification computation unit number by number of inconsistent temporal dependency

From Figure 6.4, we can see that with $L$ increasing, $\text{effe}_{\text{without}}$ increases dramatically. However, $\text{effe}_{\text{with}}$ does not increase at all with $L$ increasing and is always 0.

Particularly, we can also see that when $L$ gets larger, $\text{effe}_{\text{without}}$ gets much larger but $\text{effe}_{\text{with}}$ remains at 0. That is to say, with temporal dependency, the ineffective verification of $\text{effe}_{\text{without}}$ and its dramatic increasing trend can be avoided. Especially, as stated in Chapters 1 and 4, in real-world grid workflow systems, grid workflows normally last a long time and consequently need a large number of upper bound constraints to control and monitor their execution in terms of time [AKMK05, BAV05]. Consequently, if we ignore temporal dependency between these upper bound constraints, the number of inconsistent temporal dependency $L$ could also be
very large. Therefore, in overall terms, we can conclude that based on temporal dependency, we can significantly improve overall temporal verification effectiveness.

### 6.3.2 Quantitative efficiency analysis

At run-time execution stage, in Scenario 8 of Figure 6.3 again, we suppose there be a checkpoint, \( a_p \), between \( a_i \) and \( a_j \), i.e. \( i_N \leq ... \leq i_2 \leq i_1 < p < j_1 \leq j_2 \leq ... \leq j_N \). We focus on previous SC upper bound constraints. The corresponding discussion for previous WC upper bound constraints is similar. Correspondingly, we suppose there be \( N \) previous SC upper bound constraints in total. We denote the total number of verification computation units based on temporal dependency and that without based on temporal dependency as follows.

- \( \text{effi}_{\text{with}} \): total number of verification computation units based on temporal dependency.
- \( \text{effi}_{\text{without}} \): total number of verification computation units without based on temporal dependency.

Then, our improvement on temporal verification efficiency based on temporal dependency is reflected by comparing \( \text{effi}_{\text{with}} \) with \( \text{effi}_{\text{without}} \).

We introduce possibility \( q \) for an activity execution not exceeding its maximum duration (\( 0 \leq q \leq 1 \)). For simplicity, we assume that each activity has the same \( q \). Then, according to Algorithm 6.2, i.e. based on temporal dependency, for upper bound constraint \( U_k \) between \( a_i \) and \( a_j \), if each activity between \( a_i \) and \( a_p \) can be completed within its maximum duration, we need not conduct any verification computation. Otherwise, we must conduct the corresponding verification and the number of verification computation units is \( j_k - j_{k-1} \). In addition, the possibility for all activities between \( a_i \) and \( a_p \) not exceeding their respective maximum durations is \( q^{(p-i)} \). Therefore, for the verification of \( U_k \), the mean number of verification
computation units is \((1 - q^{p-i_{k}}) \ast (j_k - j_{k-1})\). However, if we verify \(U_k\) directly according to Definition 3.6 without based on temporal dependency, the number of verification computation units is \(j_k - p\). Hence, we have formulas (6-12) and (6-13) below.

\[
\text{effi}_{\text{with}} = \sum_{k=1}^{N} (1 - q^{(p-i_{k})}) \ast (j_k - j_{k-1}) \tag{6-12}
\]

\[
\text{effi}_{\text{without}} = \sum_{k=1}^{N} (j_k - p) \tag{6-13}
\]

We now further analyse the two formulas. For simplicity, we assume that the number of activities between any two adjacent upper bound constraints in Scenario 8 of Figure 6.3 be the same, denoted as \(Q_2\). Then, we have formulas (6-14) and (6-15) below.

\[
\text{effi}_{\text{with}} = \sum_{k=1}^{N} (1 - q^{[(p-i_{k})+(k-1)\ast Q_2]}) \ast Q_2 \tag{6-14}
\]

\[
\text{effi}_{\text{without}} = \frac{N^2(N-1)}{2} \ast Q_2 + N \ast (j_1 - p) \tag{6-15}
\]

We now take a set of values to quantitatively illustrate \(\text{effi}_{\text{with}}\) and \(\text{effi}_{\text{without}}\). Suppose \(Q_2 = 2, j_1-p = 4, p-i_1 = 2, q = 0.9, N = 20\). The selection of these values is random and does not affect our illustration because what we want to illustrate is the trend of how \(\text{effi}_{\text{with}}\) is less than \(\text{effi}_{\text{without}}\) when \(N\) changes. According to formulas (6-14) and (6-15), with \(N\) changing, we depict corresponding \(\text{effi}_{\text{with}}\) and \(\text{effi}_{\text{without}}\) in Figure 6.5.
Figure 6.5 Change trend of mean upper bound constraint verification computation unit number by number of upper bound constraints

From Figure 6.5, we can see that when $N$ increases, both $\text{effi}_{\text{with}}$ and $\text{effi}_{\text{without}}$ also increase. However, their increase rates are quite different. $\text{effi}_{\text{with}}$ increases slowly while $\text{effi}_{\text{without}}$ increases dramatically.

Particularly, from Figure 6.5, we can see that when $N$ gets larger, $\text{effi}_{\text{without}}$ gets much greater than $\text{effi}_{\text{with}}$. Similarly, as stated in Chapters 1 and 4, in real-world grid workflow systems, grid workflows normally last a long time and consequently need a large number of upper bound constraints to control and monitor their execution in terms of time [AKMK05, BAV05]. That is to say, in real-world grid workflow systems, $N$ is normally very large. Therefore, we can conclude that based on temporal dependency, we can significantly improve overall temporal verification efficiency.
6.4 Summary

In this chapter, we have systematically investigated temporal dependency between upper bound constraints and analysed its impact on temporal verification effectiveness and efficiency. Based on the investigation and analysis, we have developed some new verification methods and algorithms. The comparison and quantitative evaluation have shown that compared to those conventional verification methods without based on temporal dependency, our new methods and algorithms can significantly improve temporal verification effectiveness and efficiency.
Chapter 7

Applying to Lower Bound and Fixed-time Constraints

As stated in Section 3.1, we took upper bound constraint as the scenario to reason about our research in Section 3.2 and Chapters 4, 5 and 6. As mentioned in Section 1.1, there are still two other types of temporal constraints, i.e. lower bound constraints and fixed-time constraints. For the purpose of completeness, in this chapter, we discuss the applicability of the research results gained in Section 3.2 and Chapters 4, 5 and 6 to lower bound and fixed-time constraints. By doing so, we can generalise our research from upper bound constraint level to general temporal constraint level so that we can say our research results in terms of better temporal verification effectiveness and efficiency are valid for a broad range of “temporal constraints” rather than only for “upper bound constraints”.

Specifically, in Section 7.1, we focus on lower bound constraints. In Section 7.2, we discuss fixed-time constraints.

7.1 Applying to lower bound constraints

We denote a lower bound constraint between \(a_i\) and \(a_j\) as \(L(a_i, a_j)\) and its value as \(l(a_i, a_j)\).

As stated in Section 3.1, the upper bound constraint between \(a_i\) and \(a_j\) is denoted
as \( U(a_i, a_j) \). According to the description of upper bound and lower bound constraints in Section 1.1, \( U(a_i, a_j) \) means that the duration between \( a_i \) and \( a_j \) must be less than or equal to it. \( L(a_i, a_j) \) means that the duration between \( a_i \) and \( a_j \) must be greater than or equal to it. Apparently, \( U(a_i, a_j) \) and \( L(a_i, a_j) \) are symmetrical to each other. As a result, corresponding research results and conclusions achieved from \( U(a_i, a_j) \) can be equally applied to \( L(a_i, a_j) \).

For example, considering the definition of SC, WC, WI and SI at build-time stage, i.e. part of the research results achieved in Section 3.2, symmetrically to Definition 3.4, we have Definition 7.1 below.

**Definition 7.1** At build-time stage, \( L(a_i, a_j) \) is said to be of

1. **SC** if \( l(a_i, a_j) \leq d(a_i, a_j) \);
2. **WC** if \( d(a_i, a_j) < l(a_i, a_j) \leq M(a_i, a_j) \);
3. **WI** if \( M(a_i, a_j) < l(a_i, a_j) \leq D(a_i, a_j) \);
4. **SI** if \( D(a_i, a_j) < l(a_i, a_j) \).

We compare Definition 3.4 with Definition 7.1 in Figure 7.1 below so that we can have an intuitive picture of the symmetry between them.

Figure 7.1 clearly shows the symmetry between Definitions 3.4 and 7.1. From top to bottom, for \( U(a_i, a_j) \), the order is SC -> WC -> WI -> SI. For \( L(a_i, a_j) \), the order is the opposite, i.e. SI -> WI -> WC -> SC. Correspondingly, Definition 7.1 can be equally derived from Definition 3.4. Similarly, due to the symmetry between upper bound constraints and lower bound constraints, other results achieved in Chapters 4, 5 and 6 for upper bound constraints can also be equally applied to lower bound constraints. That is to say, the corresponding results for lower bound constraints would be similar to those for upper bound constraints. Hence, we need not further discuss how to apply the research results achieved in Section 3.2 and Chapters 4, 5 and 6 to lower bound constraints.
7.2 Applying to fixed-time constraints

We denote a fixed-time constraint at $a_i$ as $F(a_i)$ and its value as $f(a_i)$. Regarding a series of fixed-time constraints, we denote them as $F_1, F_2, F_3$ and so forth for convenience. Accordingly, we denote their values as $f(F_1), f(F_2), f(F_3)$ and so forth. In addition, as stated in Section 3.1, a grid workflow is denoted as $gw$. We denote the expected time from which the specification of grid workflow $gw$ will come into effect as $C(gw)$. From $C(gw)$, the specification of grid workflow $gw$ can be used.

When a grid workflow is executed, it starts from the first activity, i.e. $a_1$, and then reaches each fixed-time constraint gradually during the execution. Hence, $a_1$ is
the start point of each fixed-time constraint, i.e. all fixed-time constraints have the same start point [BAV05, CY05b].

Considering a fixed-time constraint, say $F(a_i)$, according to Section 3.1, we can actually treat it as a special case of an upper bound constraint whose start activity is $a_1$ and whose end activity is $a_i$. Therefore, we can also apply the research results achieved from upper bound constraints to fixed-time constraints. However, since all fixed-time constraints have the same start point, simpler or better results can be achieved for fixed-time constraints. To find out how simpler or better, we need to further investigate how to simplify and improve the research results achieved in Section 3.2 and Chapters 4, 5 and 6 for fixed-time constraints.

Specifically, in Section 7.2.1, we discuss the definitions of SC, WC, WI and SI for fixed-time constraints based on Section 3.2. In Section 7.2.2, we investigate the assignment of fine-grained fixed-time constraints based on Chapter 4. In Section 7.2.3, we discuss sufficient and necessary checkpoint selection for fixed-time constraints based on Chapter 5. Finally in Section 7.2.4, we focus on temporal dependency between fixed-time constraints based on Chapter 6.

### 7.2.1 SC, WC, WI and SI of fixed-time constraints

We derive definitions of SC, WC, WI and SI of fixed-time constraints in Section 7.2.1.1. Then, in Section 7.2.1.2, we conduct a brief comparison with SC, WC, WI and SI of upper bound constraints.

#### 7.2.1.1 Definition of SC, WC, WI and SI

Considering a fixed-time constraint at $a_i$ in grid workflow $gw$, i.e. $F(a_i)$, the build-time start time of $F(a_i)$ is $C(gw)$ because the specification of $gw$ can only be used from $C(gw)$. Then, based on Definition 3.4, we can derive Definition 7.2 below for defining SC, WC, WI and SI of $F(a_i)$ at build-time stage [CY05c, CY06d].

**Definition 7.2**

At build-time stage, $F(a_i)$ is said to be of
(1) SC if \( D(a_i, a_j) \leq f(a_j) - C(gw) \);
(2) WC if \( M(a_i, a_j) \leq f(a_j) - C(gw) < D(a_i, a_j) \);
(3) WI if \( d(a_i, a_j) \leq f(a_j) - C(gw) < M(a_i, a_j) \);
(4) SI if \( f(a_j) - C(gw) < d(a_i, a_j) \).

At run-time instantiation stage, grid workflow instances are enacted. Accordingly, we will get the absolute start time of a grid workflow, i.e. \( S(a_1) \). Then, based on Definition 3.5, we can derive Definition 7.3 below for defining SC, WC, WI and SI of \( F(a_i) \) at run-time instantiation stage [CY05c, CY06d].

**Definition 7.3**
At run-time instantiation stage, \( F(a_i) \) is said to be of

(1) SC if \( D(a_1, a_i) \leq f(a_i) - S(a_1) \);
(2) WC if \( M(a_1, a_i) \leq f(a_i) - S(a_1) < D(a_1, a_i) \);
(3) WI if \( d(a_1, a_i) \leq f(a_i) - S(a_1) < M(a_1, a_i) \);
(4) SI if \( f(a_i) - S(a_1) < d(a_1, a_i) \).

At run-time execution stage, at checkpoint \( a_p \) which is before or at \( a_i (p \leq i) \), based on Definition 3.6, we can derive Definition 7.4 below for defining SC, WC, WI and SI of \( F(a_i) \) at run-time execution stage [CY05c, CY06d].

**Definition 7.4**
At run-time execution stage, at checkpoint \( a_p \) which is before or at \( a_i (p \leq i) \), \( F(a_i) \) is said to be of

(1) SC if \( R(a_1, a_p) + D(a_{p+1}, a_i) \leq f(a_i) - S(a_1) \);
(2) WC if \( R(a_1, a_p) + M(a_{p+1}, a_i) \leq f(a_i) - S(a_1) < R(a_1, a_p) + D(a_{p+1}, a_i) \);
(3) WI if \( R(a_1, a_p) + d(a_{p+1}, a_i) \leq f(a_i) - S(a_1) < R(a_1, a_p) + M(a_{p+1}, a_i) \);
(4) SI if \( f(a_i) - S(a_1) < R(a_1, a_p) + d(a_{p+1}, a_i) \).

7.2.1.2 Comparing with SC, WC, WI and SI of upper bound constraints

Compared with Definition 3.4, 3.5 and 3.6, corresponding expressions in Definitions 7.2, 7.3 and 7.4 start from \( a_j \). This would be more intuitive, and easier for us to track fixed-time constraints at run-time execution stage. Besides, when we verify SC, WC,
WI and SI, we could utilise previous verification computation to reduce later
computation because all fixed-time constraints have the same start point and
consequently have some activities in common. However, according to Section 3.2,
different upper bound constraints may have different start points. Consequently, they
may not have activities in common. That is to say, we may not be always able to
utilise previous verification of SC, WC, WI and SI for later verification. Therefore,
verification of SC, WC, WI and SI for fixed-time constraints would be more
efficient that that for upper bound constraints.

7.2.2 Assigning fine-grained fixed-time constraints

We discuss the process of assigning fine-grained fixed-time constraints in Section
7.2.2.1. In Section 7.2.2.2, we explain why the assigning process is simpler than that
for assigning fine-grained upper bound constraints.

7.2.2.1 Assigning process

Similar to Section 4.1, we consider one fixed-time constraint among several user-
defined coarse-grained ones at build-time stage. We denote it as $F$. Similar to
Section 4.1, we focus on SC only. The corresponding discussion for WC is similar.
Accordingly, we can suppose that $F$ be of SC. Suppose $F$ be at $a_T$, then $F$ covers $T$
activities from $a_1$ to $a_T$. We suppose there be $N$ activities where temporal violations
often happen. We denote these activities as $a_{j_1}, a_{j_2}, \ldots, a_{j_N}$. Correspondingly, we
need to set $N$ fine-grained fixed-time constraints respectively at these activities. We
denote them as $F_1, F_2, \ldots, F_N$, i.e. $F_i$ at $a_{j_i} (i = 1, 2, \ldots, N)$. We also denote their
values as $f(F_1), f(F_2), \ldots, f(F_N)$.

Now we discuss how to obtain the values of $F_1, F_2, \ldots, F_N$. We allocate the
time redundancy of $F$, i.e. $[f(F) - C(gw)] - D(a_1, a_T)$, to the activities covered by $F_1,$
$F_2, \ldots, F_N$. Since $F_1, F_2, \ldots, F_N$ have the same start point $a_1$, they are nested
one after the other. Accordingly, the activities covered by $F_1, F_2, \ldots, F_N$ are just
$a_1, a_2, \ldots, a_{j_N}$. Among them, we first sort all $D(a_{j_s}) - M(a_{j_s}) (s = 1, 2, 3, \ldots, j_N)$ in
ascending order to get a sorted list. We denote the list as $L$ and the items in $L$ as $L_i,$
If \( D(a_s) - M(a_s) \) is ranked number \( k \) in \( L \), i.e. \( L_k \), then we propose formula (7-1) below to allocate \([f(F) - C(gw)] - D(a_1, a_T)\) to each of \( j_N \) activities [CY06c]. We denote the time quota allocated to \( a_s \) as \( TQ(a_s) \).

\[
TQ(a_s) = \frac{L_{j_N-k+1}}{\sum_{i=1}^{j_N} [D(a_i) - M(a_i)]} \quad (1 \leq k \leq j_N)
\]

(7-1)

The relationship between \( L_k \) and \( L_{M+k+1} \) is similar to that depicted in Figure 4.1. The explanation for formula (7-1) is similar to that for formula (4-1) in Section 4.1.1.

After we allocate \([f(F) - C(gw)] - D(a_1, a_T)\) to activities covered by \( F_1, F_2, \ldots \) and \( F_N \), each activity \( a_s \) (\( s = 1, 2, 3, \ldots, j_N \)) will carry a time quota. We can then derive the values of \( F_1, F_2, \ldots \) and \( F_N \). Considering \( F_i \), we derive its value by formula (7-2) below.

\[
f(F_i) = \sum_{s=1}^{j_N} [TQ(a_s) + D(a_s)] \quad (i = 1, 2, 3, \ldots, N)
\]

(7-2)

The explanation for formula (7-2) is similar to that for formula (4-2) in Section 4.1.1, hence omitted here.

The relationship between \( F_1, F_2, \ldots, F_N \) and \( F \) are depicted in Figure 7.2 below.
The proof for applicability of formulas (7-1) and (7-2) is similar to that for formulas (4-1) and (4-2) in Section 4.1.1, hence omitted.

### 7.2.2.2 Comparing with assignment of fine-grained upper bound constraints

At build-time stage, to assign fine-grained upper bound constraints, we can see from Section 4.1 that we need to collect all activities covered by different time slots. Then, we can allocate the time redundancy to them in order to achieve the values of fined-grained upper bound constraints. This is because different upper bound constraints may have different start points and consequently may cover different activities. However, for fixed-time constraints, since all of them have the same start point, we can immediately know all activities involved in them. They are the activities covered by the last fined-grained fixed-time constraint, i.e. $F_N$ in Figure 7.2. Clearly, the process for assigning fine-grained fixed-time constraints is simpler than that for assigning fine-grained upper bound constraints.

### 7.2.3 Selecting sufficient and necessary checkpoints for fixed-time constraint verification

In Section 7.2.3.1, we discuss the sufficient and necessary checkpoint selection process for fixed-time constraints. Then, in Section 7.2.3.2, we conduct a brief comparison with upper bound constraints.
### 7.2.3.1 Sufficient and necessary checkpoint selection for fixed-time constraints verification

Similar to Chapter 5, when grid workflow execution arrives at an activity, before the execution of the activity, all previous fixed-time constraints would be of either SC or WC. Hence, checkpoint selection actually focuses on selecting checkpoints for verifying previous SC and WC fixed-time constraints to check their current consistency. Accordingly, based on Definitions 5.1, 5.2, 5.3 and 5.4, we derive Definitions 7.5, 7.6, 7.7 and 7.8 below for SC and WC minimum time redundancy of fixed-time constraints [CY06b]. The former is for SC fixed-time constraints while the later is for WC fixed-time constraints. At activity point $a_p$ at run-time execution stage, we consider fixed-time constraints $F(a_i)$ and $F(a_j)$ ($p < i, p < j$).

**Definition 7.5 (SC Time Redundancy)**

At activity point $a_p$ ($p < i$), let $F(a_i)$ be of SC. Then, SC time redundancy of $F(a_i)$ at $a_p$ is defined as $[f(a_i) - S(a_i)] - [R(a_i, a_p) + D(a_{p+1}, a_i)]$ and is denoted as denoted as $TR_{SC}(F(a_i), a_p)$, i.e.

$$TR_{SC}(F(a_i), a_p) = [f(a_i) - S(a_i)] - [R(a_i, a_p) + D(a_{p+1}, a_i)]$$  \hspace{1cm} (7-9)

**Definition 7.6 (WC Time Redundancy)**

At activity point $a_p$ ($p < j$), let $F(a_j)$ be of WC. Then, the WC time redundancy of $F(a_j)$ at $a_p$ is defined as $[f(a_j) - S(a_j)] - [R(a_j, a_p) + M(a_{p+1}, a_j)]$ and is denoted as $TR_{WC}(FTC(a_j), a_p)$, i.e.

$$TR_{WC}(FTC(a_j), a_p) = [f(a_j) - S(a_j)] - [R(a_j, a_p) + M(a_{p+1}, a_j)]$$  \hspace{1cm} (7-10)

**Definition 7.7 (Minimum SC Time Redundancy)**

Let $F_1, F_2, ..., F_N$ be $N$ SC fixed-time constraints and all of them cover activity point $a_p$. Then, at $a_p$, minimum SC time redundancy is defined as the minimum of all SC time redundancies of $F_1, F_2, ..., F_N$, and is denoted as $MTR_{SC}(a_p)$ ($MTR$: Minimum Time Redundancy), i.e.

$$MTR_{SC}(a_p) = \text{Min} \{ TR_{SC}(F_s, a_p)| s = 1, 2, ..., N \}$$  \hspace{1cm} (7-11)
Definition 7.8 (Minimum WC Time Redundancy)

Let $F_1, F_2, \ldots, F_N$ be $N$ WC fixed-time constraints and all of them cover activity point $a_p$. Then, at $a_p$, minimum WC time redundancy is defined as the minimum of all WC time redundancies of $F_1, F_2, \ldots, F_N$, and is denoted as $MTR_{WC}(a_p)$, i.e.

$$MTR_{WC}(a_p) = \min \{ TR_{WC}(F_s, a_p) | s = 1, 2, \ldots, N \}$$

(7-12)

Obviously, at $a_{p-1}$ or just before the execution of $a_p$, minimum SC and WC time redundancies are $MTR_{SC}(a_{p-1})$ and $MTR_{WC}(a_{p-1})$ respectively. Similar to the discussion in Section 5.1.2, we normally have $M(a_p) + MTR_{WC}(a_{p-1}) < D(a_p) + MTR_{SC}(a_{p-1})$.

We now move on to checkpoint selection based on Minimum SC and WC time redundancy. As stated in Chapter 5, the process of selecting sufficient and necessary checkpoints consists of two sub processes.

- One is that we dynamically obtain $MTR_{SC}$ and $MTR_{WC}$ along grid workflow execution:
  The dynamic obtaining of $MTR_{SC}$ and $MTR_{WC}$ of fixed-time constraints along grid workflow execution could be derived from DOMTR in Section 5.1.3 by taking into consideration the fact that all fixed-time constraints have the same start activity $a_1$. We denote it as $F$-DOMTR and describe it below.

- The other is that at an activity, say $a_p$, we decide whether we should take it as a checkpoint:
  The decision-making approach is similar to $CDA(a_p)$ discussed in Section 5.3.1 because the decision-making process has nothing to do with the fact that all fixed-time constraints have the same start activity $a_1$. We denote it as $F$-$CDA(a_p)$. Based on $CDA(a_p)$, $F$-$CDA(a_p)$ is: At activity $a_p$, if $R(a_p) > D(a_p) + MTR_{SC}(a_{p-1})$, we take it as a checkpoint for verifying SC, WC, WI & SI of all previous SC fixed-time constraints, and for verifying WC, WI & SI of all previous WC fixed-time constraints. If $M(a_p) + MTR_{WC}(a_{p-1}) < R(a_p) \leq D(a_p)$
+ $MTR_{SC}(a_{p-1})$, we take $a_p$ as a checkpoint for verifying SC, WC, WI & SI of all previous WC fixed-time constraints only. If $R(a_p) \leq M(a_p) + MTR_{WC}(a_{p-1})$, we do not take $a_p$ as a checkpoint.

**Working steps of F-DOMTR:**

**At run-time instantiation stage** (setting up initial values)

**Step 1.**
During the fixed-time constraint verification process, for each SC fixed-time constraint, say $F(a_i)$, compute $f(a_i) - S(a_1) - D(a_1, a_i)$, denoted as SC time difference. For each WC fixed-time constraint, say $F(a_j)$, compute $f(a_j) - S(a_1) - M(a_1, a_j)$, denoted as WC time difference. All time differences can be obtained by utilising corresponding verification computation results from the build-time stage.

**Step 2.**
At $a_1$, the start point of all SC and WC fixed-time constraints, compare all SC time differences to derive the minimum one. Denote it as $SMTD_{SC-init}(a_1)$ (SMTD: Minimum Time Difference at Start activity; *init*: initial). Similarly, obtain the minimum WC time difference and denote it as $SMTD_{WC-init}(a_1)$.

**Step 3.**
At the end activity of each SC fixed-time constraint, derive another minimum SC time difference. But this one is different from the one derived in Step 2. For example, at $a_i$ of SC $F(a_i)$, compare all SC time differences of those SC fixed-time constraints which cover $a_i$, but do not end at $a_i$. Denote the minimum one as $EMTD_{SC-init}(a_i)$ (EMTD: Minimum Time Difference at End activity; *init*: initial). Similarly, at the end activity of each WC fixed-time constraint, derive another minimum WC time difference and, at $a_j$ of WC $F(a_j)$, denote it as $EMTD_{WC-init}(a_j)$.

**Step 4.**
For each activity, say $a_r$, which is not covered by any SC or WC fixed-time constraints, set $MTR_{SC}(a_r)$ and $MTR_{WC}(a_r)$ to the biggest possible float number of the system.

115
At run-time execution stage (computing $MTR_{SC}$ and $MTR_{WC}$)

Step 5.
At $a_i$, $MTR_{SC}(a_i) = SMTD_{SC\text{-}init}(a_i) - [R(a_i) - D(a_i)];$ $MTR_{WC}(a_i) = SMTD_{WC\text{-}init}(a_i) - [R(a_i) - M(a_i)].$

Step 6.
When grid workflow execution arrives at an activity, say $a_p$, which is covered by some SC or WC fixed-time constraints, but is not an end activity of any SC or WC fixed-time constraint, after the execution of $a_p$, $MTR_{SC}(a_p) = MTR_{SC}(a_{p-1}) - [R(a_p) - D(a_p)]$ and $MTR_{WC}(a_p) = MTR_{WC}(a_{p-1}) - [R(a_p) - M(a_p)].$

Step 7.
Along grid workflow execution, we now discuss how to obtain new $MTR_{SC}$ and $MTR_{WC}$ when grid workflow execution arrives at an end activity of an SC or WC fixed-time constraint. We focus on new $MTR_{SC}$. For new $MTR_{WC}$, the corresponding discussion is similar. Suppose now grid workflow execution arrives at the end activity of SC $F(a_i)$, namely $a_i$, denote the SC fixed-time constraint corresponding to $MTR_{SC}(a_{i-1})$ as $F(MTR_{SC}(a_{i-1}))$.

Step 7.1.
If $a_i$ is not the end activity of $F(MTR_{SC}(a_{i-1}))$, i.e. $F(MTR_{SC}(a_{i-1}))$ is different from $F(a_i)$, obtain $MTR_{SC}(a_{i})$ according to Step 6 as $F(MTR_{SC}(a_{i-1}))$ is still valid.

Step 7.2.
If $a_i$ is the end activity of $F(MTR_{SC}(a_{i-1}))$, i.e. $F(MTR_{SC}(a_{i-1}))$ is just $F(a_i)$, then, $MTR_{SC}(a_i)$ can also be obtained by Step 6. However, such $MTR_{SC}(a_i)$ cannot be used after the execution of $a_i$ because $F(MTR_{SC}(a_{i-1}))$ will be invalid. For example, we cannot compute $MTR_{SC}(a_{i+1})$ based on such $MTR_{SC}(a_i)$. Hence, after execution of $a_i$, we need to set a new $MTR_{SC}(a_i)$ to replace such $MTR_{SC}(a_i)$. The new $MTR_{SC}(a_i)$ depends on two situations: 1) there are no other SC fixed-time constraints after $a_i$; and 2) there are some other SC fixed-time constraints after $a_i$. 
Step 7.2.1.
For 1), the new $MTR_{SC}(a_i)$ is the same as that obtained by Step 6.

Step 7.2.2.
For 2), the new $MTR_{SC}(a_i) = EMTD_{SC-init}(a_i) - \sum_{s=1}^{i} [R(a_s) - D(a_s)]$.

Step 8.
When grid workflow execution arrives at an activity that is not covered by any SC or WC fixed-time constraints, we do nothing and simply keep the initial values set by Step 4.

Step 9.
Along grid workflow execution, repeat all or some of Steps 5, 6, 7 and 8 when applicable.

Combining F-DOMTR with $F-CDA(a_p)$ together, we can derive a checkpoint selection strategy for fixed-time constraint verification. We denote it as $F-CSS_{MTR}$. Based on Algorithm 5.1, we depict the main part of $F-CSS_{MTR}$ in Algorithm 7.1 below. We can also prove that the checkpoints selected by $F-CSS_{MTR}$ are sufficient and necessary for fixed-time constraint verification. The corresponding proofs are similar to those in Theorems 5.4 and 5.5 in Section 5.3.2, hence omitted here.

<table>
<thead>
<tr>
<th>Input</th>
<th>Maximum, minimum and mean durations of all activities; all SC and WC fixed-time constraints.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td><em>True</em> or <em>False</em> as an appropriate checkpoint.</td>
</tr>
<tr>
<td>Step 1</td>
<td>At run-time instantiation stage, conduct F-DOMTR to set up some initial values based on fixed-time constraint verification computation results.</td>
</tr>
</tbody>
</table>

| 1.1 Execute Steps 1 and 2 of F-DOMTR to obtain $SMTD_{SC-init}(a_t)$ and $SMTD_{WC-init}(a_t)$. |
| 1.2 Execute Step 3 of F-DOMTR to obtain $EMTD_{SC-init}$ and... |
**EMTD**\(_{WC-init}\) for every end activity of each SC or WC fixed-time constraint.

1.3 Execute Step 4 of F-DOMTR to set the biggest float number of the system to \(MTR_{SC}\) and \(MTR_{WC}\) of each activity that is not covered by any SC or WC fixed-time constraints.

<table>
<thead>
<tr>
<th>Step 2</th>
<th>At run-time execution stage, conduct F-DOMTR to obtain (MTR_{SC}(a_{p-1})) and (MTR_{WC}(a_{p-1})) when grid workflow execution arrives at (a_{p-1}).</th>
</tr>
</thead>
</table>

2.1 If \(a_{p-1}\) is \(a_1\), execute Step 5 of F-DOMTR to obtain \(MTR_{SC}(a_1)\) and \(MTR_{WC}(a_1)\).

2.2 If \(a_{p-1}\) is an intermediate activity of some SC and/or WC upper bound constraints, then execute Step 6 of F-DOMTR to obtain \(MTR_{SC}(a_{p-1})\) and \(MTR_{WC}(a_{p-1})\).

2.3 If \(a_{p-1}\) is an end activity of some SC and/or WC upper bound constraints, then execute Step 7 of F-DOMTR including Steps 7.1, 7.2, 7.2.1, 7.2.2 to obtain \(MTR_{SC}(a_{p-1})\) and \(MTR_{WC}(a_{p-1})\).

2.4 If \(a_{p-1}\) is not covered by any SC or WC upper bound constraints, then execute Step 8 of F-DOMTR to obtain \(MTR_{SC}(a_{p-1})\) and \(MTR_{WC}(a_{p-1})\).

<table>
<thead>
<tr>
<th>Step 3</th>
<th>At run-time execution stage, call <strong>F-CDA</strong>(<em>{SC})(</em>{ap}) to decide whether (a_p) should be selected as an appropriate checkpoint when grid workflow execution arrives at (a_p).</th>
</tr>
</thead>
</table>

3.1 Call **F-CDA**\(_{SC}\)\(_{ap}\) to compare \(D(a_p) + MTR_{SC}(a_{p-1})\) and \(M(a_p) + MTR_{WC}(a_{p-1})\) with \(R(a_p)\) so that we can decide whether \(a_p\) should be selected as an appropriate checkpoint.

3.2 According to 3.1, output “True” or “False” for selecting \(a_p\) as a checkpoint.

**Algorithm 7.1 Checkpoint selection process of F-CSS\(_{MTR}\)**
### 7.2.3.2 Comparing with upper bound constraints

As stated in Section 7.2.3.1, the process of selecting sufficient and necessary checkpoints consists of two sub processes. One is that we dynamically obtain \( MTR_{SC} \) and \( MTR_{WC} \) along grid workflow execution. The other is that at an activity, say \( a_p \), we decide whether we should take it as a checkpoint. Correspondingly, to compare the checkpoint selection process of fixed-time constraints with that of upper bound constraints, we compare such two sub processes respectively. Namely, we compare F-DOMTR with DOMTR on one hand. And we compare \( F-CDA(a_p) \) with \( CDA(a_p) \) on the other hand.

For F-DOMTR with DOMTR, from Section 5.1.3, we can see that DOMTR computes \( MTR_{SC} \) and \( MTR_{WC} \) at three types of activities: start activities, intermediate activities and end activities of upper bound constraints. Especially, when grid workflow execution is entering into a new upper bound constraint, much computation needs to be conducted. However, F-DOMTR does not need to allow for this case. This is because all fixed-time constraints have the same start point which is just \( a_1 \). Therefore, F-DOMTR is simpler than DOMTR.

For \( F-CDA(a_p) \) with \( CDA(a_p) \), we can see that they are very similar. This is because they have nothing to do with the fact that all fixed-time constraints have the same start point \( a_1 \).

As we can see from Algorithms 5.1 and 7.1, DOMTR is the key component of the sufficient and necessary checkpoint selection process for upper bound constraints and F-DOMTR is the key component of the corresponding process for fixed-time constraints. Therefore, in overall terms, we can say that the sufficient and necessary checkpoint selection process for fixed-time constraints is simpler than that for upper bound constraints. That is to say, in terms of sufficient and necessary checkpoint selection, simpler research results can be achieved for fixed-time constraints than those for upper bound constraints.
7.2.4 Temporal dependency between fixed-time constraints

In Section 7.2.4.1, we discuss temporal dependency between fixed-time constraints. Then, in Section 7.2.4.2, we conduct a brief comparison with upper bound constraints.

7.2.4.1 Temporal dependency between fixed-time constraints

Similar to Section 6.1, when grid workflow execution arrives at an activity, before the execution of the activity, all previous fixed-time constraints would be of either SC or WC. Correspondingly, temporal dependency between fixed-time constraints consists of SC temporal dependency and WC temporal dependency. The former is based on SC state of fixed-time constraints while the later is based on WC state.

In addition, similar to Chapter 6 which discusses temporal dependency between upper bound constraints, at build-time stage we discover SC and WC temporal dependency between fixed-time constraints. At run-time instantiation stage, we will have the absolute start time of a grid workflow, i.e. $S(a_j)$. However, SC or WC temporal dependency is a relevant concept between fixed-time constraints. Consequently, both of them have nothing to do with $S(a_j)$. Hence, the corresponding discussion is the same as that of build-time stage. At run-time execution stage, we discuss fixed-time constraint verification based on SC and WC temporal dependency.

At build-time stage, considering two fixed-time constraints $F_1$, $F_2$ (see Figure 7.2), based on the discussion in Section 6.1.1 and Definitions 6.1 and 6.2, we can derive Definitions 7.9 and 7.10 below for defining SC and WC temporal dependency between $F_1$ and $F_2$.

**Definition 7.9 (SC Temporal Dependency)**

Let $F_1$ and $F_2$ be two fixed-time constraints at $a_{j_1}$ and $a_{j_2}$ respectively ($j_1 < j_2$) (see Figure 7.2). Then, with $D(a_{j_1+1}, a_{j_2}) \leq f(F_2) - f(F_1)$, SC temporal dependency between $F_1$ and $F_2$ is defined as consistent.
Definition 7.10 (WC Temporal Dependency)

Let $F_1$ and $F_2$ be two fixed-time constraints at $a_{j_1}$ and $a_{j_2}$ respectively ($j_1 < j_2$) (see Figure 7.2). Then, with $M(a_{j_1}, a_{j_2}) \leq f(F_2) - f(F_1)$, WC temporal dependency between $F_1$ and $F_2$ is defined as consistent.

For a series of fixed-time constraints $F_1, F_2, \ldots, F_N$ (see Figure 7.2), based on Theorem 6.1, we can derive Theorem 7.1 below. According to Theorem 7.1, SC or WC temporal dependency between $F_1, F_2, \ldots, F_N$ can be translated into that between two fixed-time constraints such as between $F_1$ and $F_2$.

Theorem 7.1

Let $F_1, F_2, \ldots, F_N$ be $N$ fixed-time constraints (see Figure 7.2) at $a_{j_1}, a_{j_2}, \ldots, a_{j_N}$ respectively ($j_1 < j_2 < \ldots < j_N$). Then,

1) if SC temporal dependency between any two adjacent fixed-time constraints $F_k$ and $F_{k+1}$ is consistent ($1 \leq k \leq N-1$), SC temporal dependency between any two non-adjacent fixed-time constraints must also be consistent;

2) if WC temporal dependency between any two adjacent fixed-time constraints $F_k$ and $F_{k+1}$ is consistent ($1 \leq k \leq N-1$), WC temporal dependency between any two non-adjacent fixed-time constraints must also be consistent.

Proof:

The proof is similar to that for Theorem 6.1, hence omitted.

Regarding build-time verification of fixed-time constraints, on one hand, we need to verify fixed-time constraints according to Definition 7.2. On the other hand, we need to verify SC and WC temporal dependency according to Definitions 7.9 and 7.10 and Theorem 7.1. The overall verification process would be similar to that of upper bound constraints conducted in Section 6.2.1 because both of the verification processes follow similar definitions and theorems. Hence, we simply omit the verification process for fixed-time constraints here and we can refer to Section 6.2.1 for details.
Now, we move on to run-time execution stage. We discuss the verification of fixed-time constraints based on their SC and WC temporal dependency. As shown in Figure 7.2, all fixed-time constraints have the same start point \( a_1 \). Therefore, Figure 7.2 is actually a special case of Scenario 9 in Figure 6.3 by replacing the common start point with \( a_1 \). Hence, based on Corollary 6.2, we can derive Theorem 7.2 below.

**Theorem 7.2**

Let \( F_1, F_2, \ldots, F_N \) be \( N \) fixed-time constraints (see Figure 7.2) at \( a_{j_1}, a_{j_2}, \ldots, a_{j_N} \) respectively (\( j_1 < j_2 < \ldots < j_N \)). Then, at checkpoint \( a_p \) between \( a_1 \) and \( a_{j_k} \),

1) if \( F_k \) is of SC, any fixed-time constraint \( F_s \) after \( F_k \) must also be of SC (\( k < s \leq N \));

2) if \( F_k \) is of WC, any fixed-time constraint \( F_s \) after \( F_k \) must also be of WC or even SC (\( k < s \leq N \)).

**Proof:**

The proof is similar to that for Corollary 6.2, hence omitted.

According to Theorem 7.2, when we verify a series of fixed-time constraints as shown in Figure 7.2, we verify them one after another until we meet an SC or WC one or finish all of them. Once we meet an SC or WC fixed-time constraint, we need not conduct further verification for any other fixed-time constraints after it. This verification process is similar to that for upper bound constraint verification in Scenario 9 of Figure 6.3.

Based on Algorithm 6.2, we can derive an algorithm for fixed-time constraint verification with temporal dependency at run-time execution stage. When deriving from Algorithm 6.2, what we need to do is to take into consideration the fact that all fixed-time constraints have the same start point which is \( a_1 \). Correspondingly, the deriving process would be a process of simplifying Algorithm 6.2. Hence, we simply omit the algorithm for fixed-time constraint verification.
7.2.4.2 Comparing with upper bound constraints

We now briefly compare the temporal dependency discussion in Section 7.2.4.1 for fixed-time constraints with that in Chapter 6 for upper bound constraints.

At build-time stage, as stated in Section 7.2.4.1, the temporal dependency discussion for fixed-time constraints is similar to that for upper bound constraints in Sections 6.1 and 6.2.1 because they have similar definitions and theorems.

At run-time instantiation stage, the temporal dependency discussion for fixed-time constraints is also similar to that for upper bound constraints. This is because there is only one time difference between build-time stage and run-time instantiation stage, i.e. $S(a_1)$, but temporal dependency definition and verification have nothing to with $S(a_1)$.

At run-time execution stage, since all fixed-time constraints have the same start point $S(a_1)$, we have derived Theorem 7.2. According to Theorem 7.2, we can derive the consistency of later fixed-time constraints directly from the consistency of previous fixed-time constraints. However, for upper bound constraints, according to Theorem 6.2 in Section 6.2.2, we often need to conduct extra computation and comparison so that we can derive the consistency of succeeding upper bound constraints from the consistency of previous upper bound constraints. We may note that there is Corollary 6.2 for those upper bound constraints which have the same start point. In this case, we can derive the consistency of succeeding upper bound constraints directly from the consistency of previous upper bound constraints. However, not all upper bound constraints have the same start point. In most cases, upper bound constraints do not have the same start point [EPR99]. Therefore, the verification process for fixed-time constraints would be simpler and more efficient than that for upper bound constraints. This would further contribute to better overall temporal verification efficiency.

In overall terms, we can conclude: in comparison with the temporal dependency based upper bound constraint verification, the corresponding fixed-time constraint
verification is simpler and even better in terms of overall temporal verification efficiency.

7.3 Summary

In this Chapter, to generalise our research results achieved in Sections 3.2 and Chapters 4, 5 and 6 from upper bound constraint level to temporal constraint level, we have discussed the applicability of the research results achieved in Section 3.2 and Chapters 4, 5 and 6 for upper bound constraints to lower bound and fixed-time constraints.

For lower bound constraints, from Section 7.1, we have seen: due to the symmetric relationship between lower bound constraints and upper bound constraints, the corresponding research results for upper bound constraints can be equally applied to lower bound constraints.

For fixed-time constraints, from Sections 7.2.1, 7.2.2 and 7.2.3, we have seen: the corresponding research results for fixed-time constraints in terms of four temporal consistency states, assignment of fine-grained upper bound constraints, and checkpoint selection are simpler than those for upper bound constraints. From Section 7.2.4, we have seen: the corresponding research results for fixed-time constraints in terms of temporal dependency are simpler and better than those for upper bound constraints.

Whether the research results achieved from upper bound constraints are equally applied to lower bound constraints or simplified and improved for fixed-time constraints, the advantages of the research results in terms of better temporal verification effectiveness and efficiency can be inherited from upper bound constraints to lower bound and fixed-time constraints. Therefore, in overall terms, we can say that the research results achieved in previous chapters in terms of better temporal verification effectiveness and efficiency are valid for all types of temporal constraints.
Chapter 8

Extension to the Handling of Temporal Verification Results

Once we verify temporal constraints according to the previous chapters, we would derive the verification results, i.e. the SC, WC, WI or SI state of temporal constraints. Naturally, the next step is to handle these results. As reflected by the title of this thesis, and introduced in Chapters 1 and 2, the primary focus of this thesis involves temporal verification in particular while the handling of SC, WC, WI and SI generally belongs to the exception handling scope which is after temporal verification. That is to say, the handling of SC, WC, WI and SI is not a main stream of this thesis. However, we propose SC, WC, WI and SI in Chapter 3 for better temporal verification effectiveness and particularly temporal verification is based on them. So it would be more comprehensive if we can provide a glimpse of the handling of SC, WC, WI & SI, and give an overall picture about the advantage of introducing them in terms of temporal verification effectiveness. For this purpose, in Section 8.1, we take WC as an example to provide a glimpse of the handling. In particular, we present a method on how to utilise the potential time redundancy of succeeding activity execution to adjust WC. For SC, according to Section 3.2.3, we need not conduct any handling. For WI, simpler handling can be deployed. However, we leave the detailed discussion as future work. For SI, conventional handling can be deployed, say the approaches described in [HA00, LSKM00]. In Section 8.2, we conduct an overall quantitative evaluation to demonstrate the advantage of introducing SC, WC, WI and SI in terms of overall handling cost effectiveness which, according to Section 2.1, contributes to overall temporal verification
Similar to previous chapters, we take upper bound constraints as the scenario to conduct Sections 8.1 and 8.2. Based on Chapter 7, corresponding results can be equally applied to lower bound constraints and can be simplified and improved for fixed-time constraints.

8.1 WC adjustment

According to Section 3.1, we normally have $M(a_i) \leq D(a_i)$. Apparently, there is a time difference between $M(a_i)$ and $D(a_i)$. We define it as mean activity time redundancy. Mean activity time redundancy indicates statistically how much redundant time an activity has. It can be used to tolerate certain time deviation of activity execution.

**Definition 8.1 (Mean Activity Time Redundancy)**

The mean activity time redundancy of $a_i$ is defined as the difference between its maximum duration and mean duration, i.e. $D(a_i) - M(a_i)$.

Now, we consider a WC upper bound constraint, say $U(a_i, a_j)$. To adjust it to SC at build-time stage, according to items 1 and 2 of Definition 3.4, we need to compensate a time deficit which is $D(a_i, a_j) - u(a_i, a_j)$. At run-time instantiation stage, according to items 1 and 2 of Definition 3.5, the time deficit is also $D(a_i, a_j) - u(a_i, a_j)$. At run-time execution stage, according to items 1 and 2 of Definition 3.6, at checkpoint $a_p$ between $a_i$ and $a_j$, the time deficit is $R(a_i, a_p) + D(a_{p+1}, a_j) - u(a_i, a_j)$.

We take run-time execution stage as the example to reason about WC adjustment. The corresponding discussion for build-time and run-time instantiation stages is similar. Accordingly, we have to compensate the time deficit: $R(a_i, a_p) + D(a_{p+1}, a_j) - u(a_i, a_j)$. According to the discussion in Sections 3.2.2 and 3.2.3 about WC, our approach is to utilise potential time redundancy of succeeding activities to compensate the time deficit. Because in real world grid workflow systems, there are
often a lot of grid workflow instances in one grid workflow system, we perform WC adjustment in a statistical way. Correspondingly, we will utilise the mean activity time redundancy of succeeding activities defined in Definition 8.1 to compensate the time deficit so that WC $U(a_i, a_j)$ can be adjusted to SC.

To compensate the time deficit: $R(a_i, a_p) + D(a_{p+1}, a_j) - u(a_i, a_j)$, we allocate it to the succeeding activities between $a_{p+1}$ and $a_j$. We are not able to allocate the deficit to other activities because all activities before $a_p$ have already completed and those after $a_j$ have nothing to do with the consistency of $U(a_i, a_j)$. Considering an activity between $a_{p+1}$ and $a_j$, say $a_k$, we make the following temporary denotations which are only effective in this chapter.

- $rdq_{ij}(a_k)$ : deficit quota to be allocated to $a_k$ by $U(a_i, a_j)$ at run-time execution stage.
- $dq(a_k)$ : current deficit quota that $a_k$ is holding.

If there is only one WC upper bound constraint, i.e. $U(a_i, a_j)$, we propose formula (8-1) to derive $rdq_{ij}(a_k)$.

$$rdq_{ij}(a_k) = \left[ R(a_i, a_p) + D(a_{p+1}, a_j) - u(a_i, a_j) \right] \frac{D(a_k) - M(a_k)}{\sum_{l=p+1}^{j} [D(a_l) - M(a_l)]} \quad (p+1 \leq k \leq j)$$

(8-1)

We now further explain formula (8-1). In formula (8-1), we allocate the time deficit to $a_k$ based on the proportion of its mean time redundancy $D(a_k) - M(a_k)$ out of the overall mean time redundancy of all activities between $a_{p+1}$ and $a_j$. We can see that an activity with bigger mean time redundancy takes a larger time deficit quota. This is because the activity with a bigger mean time redundancy has more time to compensate the time deficit.
To be able to apply formula (8-1), we need to ensure \( rdq_{ij}(a_k) \leq D(a_k) - M(a_k) \). Otherwise, the available time for \( a_k \) to complete is less than its mean duration \( M(a_k) \). Therefore, statistically for most cases, the allocation will probably lead to new WC or WI or SI. This means that the allocation should not be applied. We derive Theorem 8.1 to support this point.

**Theorem 8.1**

At run-time execution stage, for WC upper bound constraint \( U(a_i, a_j) \), if we allocate the time deficit to activity \( a_k \) between \( a_{p+1} \) and \( a_j \) according to formula (8-1), then, we have: \( rdq_{ij}(a_k) \leq D(a_k) - M(a_k) \).

**Proof:**

To prove \( rdq_{ij}(a_k) \leq D(a_k) - M(a_k) \), according to formula (8-1), we only need to prove

\[
R(a_i, a_p) + D(a_{p+1}, a_j) - u(a_i, a_j) \leq \sum_{l=p+1}^{j}(D(a_l) - M(a_l)) - \sum_{l=p+1}^{j}M(a_l) = D(a_{p+1}, a_j) - M(a_{p+1}, a_j) \]

That is to say, we only need to prove \( R(a_i, a_p) + D(a_{p+1}, a_j) - u(a_i, a_j) \leq D(a_{p+1}, a_j) - M(a_{p+1}, a_j) \). Finally, we only need to prove that inequation (8-2) below holds.

\[
R(a_i, a_p) + M(a_{p+1}, a_j) \leq u(a_i, a_j) \tag{8-2}
\]

Because \( U(a_i, a_j) \) is of WC, according to item 2 of Definition 3.6, we do have

\[
R(a_i, a_p) + M(a_{p+1}, a_j) \leq u(a_i, a_j) .
\]

Thus, in overall terms, the theorem holds.

If there are a series of WC upper bound constraints, we may need to conduct multiple allocations. This is because those WC upper bound constraints may cover some activities in common. For such common activities, there will be multiple time deficit allocations. Therefore, we need to investigate how to conduct the time deficit allocation for a series of WC upper bound constraints together. Based on formula (8-
1) and Theorem 8.1, we derive a systematic time deficit allocation approach for a series of WC upper bound constraints. For convenience, we name the approach as RTDA (Run-time Time Deficit Allocation).

The detailed working process of RTDA is depicted as follows.

Suppose now we are ready to allocate the time deficit of WC $U(a_i, a_j)$ to $a_k$. After we derive $rdq_{ij}(a_k)$ by using formula (8-1), we compare $rdq_{ij}(a_k)$ with $dq(a_k)$. If $dq(a_k)$ is less, we replace the value of $dq(a_k)$ with $rdq_{ij}(a_k)$. Otherwise, we do nothing.

We now further explain RTDA. In RTDA, if $dq(a_k) < rdq_{ij}(a_k)$, we will set $rdq_{ij}(a_k)$ to $dq(a_k)$. This is because $dq(a_k) < rdq_{ij}(a_k)$ means that previous allocations are not sufficient for current WC adjustment. However, $rdq_{ij}(a_k)$, there will be more time for previous WC upper bound constraints to switch to SC. Therefore, when $dq(a_k) < rdq_{ij}(a_k)$, we replace the value of $dq(a_k)$ with that of $rdq_{ij}(a_k)$. If $rdq_{ij}(a_k) \leq dq(a_k)$, then the previous allocations would be enough for current WC adjustment. Hence, we simply take the previous allocated $dq(a_k)$ without changing it.

To be able to apply RTDA, on one hand, we must prove that all allocations are enough for their respective WC upper bound constraints to switch to SC. Therefore, Theorem 8.2 below is presented. On the other hand, similar to $rdq_{ij}(a_k)$, we also need to ensure $dq(a_k) \leq D(a_k) - M(a_k)$. Theorem 8.3 below supports this point.

**Theorem 8.2**

At run-time execution stage, given a series of WC upper bound constraints, if we allocate their time deficits to the succeeding activities according to formula (8-1) and RTDA, the final allocation results are enough for all WC upper bound constraints to switch to SC.
Proof:
For any succeeding activity, say $a_k$, of any WC upper bound constraint, say $U(a_i, a_j)$, after all allocations are conducted, according to the allocating process of RTDA, we have inequation (8-3) below.

$$rdq_{ij}(a_k) \leq dq(a_k) \quad (8-3)$$

Inequation (8-3) means that at $a_k$, by taking $dq(a_k)$, $U(a_i, a_j)$ can get more time to switch to SC than with its own deficit allocation. Since $U(a_i, a_j)$ can switch to SC even only based on its own deficit allocation, based on multiple allocations, $U(a_i, a_j)$ will have more chance to switch to SC.

Thus, the theorem holds.

Theorem 8.3
At run-time execution stage, given a series of WC upper bound constraints, if we allocate their time deficits to succeeding activities according to formula (8-1) and RTDA, then, for any succeeding activities, say $a_k$, we have: $dq(a_k) \leq D(a_k) - M(a_k)$.

Proof:
According to RTDA, $dq(a_k)$ is equal to the maximum deficit quota allocated to $a_k$. According to Theorem 8.1, any deficit quota allocated to $a_k$ is less than or equal to $D(a_k) - M(a_k)$. Therefore, the maximum quota must also be less than or equal to $D(a_k) - M(a_k)$.

Thus, the theorem holds.

In overall terms, based on RTDA, we can derive an algorithm for adjusting a series of WC upper bound constraints to SC. We depict it in Algorithm 8.1 below.
### Input
- ArrayWU: an array of all current WC upper bound constraints which cover checkpoint \(a_p\);
- Maximum and mean durations of all activities involved in ArrayWU;

### Output
- WC adjustment report for all current WC upper bound constraints

### Step 1
Set up initial values

For each of the activities after \(a_p\) and covered by one or more WC upper bound constraints from ArrayWU, say \(a_s\), set its initial value of \(dq(a_s)\) to 0.

### Step 2
Consider current WC upper bound constraint from ArrayWU

1. Select current WC upper bound constraint from ArrayWU to \(U(a_i, a_j)\) \((i \leq p \leq j)\);
2. For an activity between \(a_{p+1}\) and \(a_j\), say \(a_k\), compute \(rdq_{ij}(a_k)\) as follows, i.e. formula (8-1);

\[
rdq_{ij}(a_k) = \left[ R(a_i, a_p) + D(a_{p+1}, a_j) - u(a_i, a_j) \right] \frac{D(a_k) - M(a_k)}{\sum_{i=p+1}^{j} [D(a_i) - M(a_i)]}
\]

### Step 3
Allocate \(rdq_{ij}(a_k)\) to \(a_k\)

If \((dq(a_k) < rdq_{ij}(a_k))\) then

\[dq(a_k) = rdq_{ij}(a_k)\]

End if

Output “\(dq(a_k)\)”;
Algorithm 8.1 Time deficit allocation at checkpoint $a_p$ at run-time execution stage for a series of WC upper bound constraints

8.2 Quantitative evaluation of SC, WC WI and SI

In this section, we conduct a quantitative analysis to present an overall picture about the advantage of SC, WC, WI and SI. In overall terms, according to Sections 3.2.4 and 8.1, the advantage is better handling cost effectiveness than with the conventional two states of CC and CI. This is because WC can be adjusted without triggering any exception handling while WI can be handled by more economical exception handling. However, with the conventional two states, WC, WI and SI are covered by CI and are handled by the same more costly exception handling.

For simplicity, we only consider the main handling cost which is spent on the activities such as compensation for executed activities [HA00, KD00, LLGS+05]. First, we make the following temporary denotations which are for this section only:

- $gw$: a grid workflow concerned.
- $C_k(gw)$: handling cost of $a_k$.
- $N(gw)$: number of WC.
- $M(gw)$: number of WI.
- $L(gw)$: number of SI.
- $Q_i(gw)$: number of activities addressed by the exception handling conducted
by the conventional work for the $i^{th}$ WC or WI or SI.

- $P_i(gw)$: number of activities addressed by the exception handling conducted by our work for the $i^{th}$ WI.

- $PC_{total}(gw)$: total handling cost of the conventional work based on conventional two states CC and CI.

- $OC_{total}(gw)$: total handling cost of our research based on our four states SC, WC, WI and SI.

According to the discussion in Sections 3.2.2 and 3.2.3, $P_i(gw)$ is less than $Q_i(gw)$ and statistically we have formulas (8-4) and (8-5) below.

$$PC_{total}(gw) = \sum_{i=1}^{N_{(gw)}} \sum_{k=1}^{Q_i(gw)} C_k(gw) + \sum_{j=1}^{M_{(gw)}} \sum_{s=1}^{Q_j(gw)} C_s(gw) + \sum_{l=1}^{L_{(gw)}} \sum_{q=1}^{Q_l(gw)} C_q(gw) \quad (8-4)$$

$$OC_{total}(gw) = \sum_{i=1}^{M_{(gw)}} \sum_{m=1}^{P_i(gw)} C_m(gw) + \sum_{l=1}^{L_{(gw)}} \sum_{n=1}^{Q_l(gw)} C_n(gw) \quad (8-5)$$

From formulas (8-4) and (8-5), we can see that $OC_{total}(gw)$ is less than or equal to $PC_{total}(gw)$. Now we further investigate to what extent $OC_{total}(gw)$ is less than or equal to $PC_{total}(gw)$. We denote the difference between $PC_{total}(gw)$ and $OC_{total}(gw)$ as $DIFF_{total}(gw)$. And we have formula (8-6) below.

$$DIFF_{total}(gw) = \sum_{i=1}^{N_{(gw)}} \sum_{k=1}^{Q_i(gw)} C_k(gw) + \sum_{j=1}^{M_{(gw)}} \sum_{s=1}^{Q_j(gw)} C_s(gw) - \sum_{i=1}^{M_{(gw)}} \sum_{m=1}^{P_i(gw)} C_m(gw) \quad (8-6)$$

To obtain an overall analysis, we analyse $DIFF_{total}(gw)$ in a statistical way. Therefore, we replace the corresponding variables in formula (8-6) with their respective mean values which can be achieved based on the past execution history.
We denote the corresponding mean values as follows:

- $C$: mean value of $C_k(gw)$.
- $N$: mean value of $N(gw)$.
- $M$: mean value of $M(gw)$.
- $L$: mean value of $L(gw)$.
- $\text{DIFF}_{total}$: mean value of $\text{DIFF}_{total}(gw)$.

In addition, according to [Liu98, SK01], statistically $Q_j(gw)$ and $P_i(gw)$ have a type of stochastic distribution. Since we discuss $\text{DIFF}_{total}(gw)$ in a statistical way, we take the mean distribution without losing generality as for other stochastic distributions, the final conclusions would be similar. Correspondingly, we have $Q_j(gw) = X * A_j$, $P_i(gw) = Y * B_i$. $A_j$ stands for the number of activities covered by the $j^{th}$ WC. $B_i$ stands for the number of activities covered by the $i^{th}$ WI. $X$ and $Y$ are mean weights that depend on the mean system load, availability of grid services and the mean time deficit incurred by grid workflow execution ($0 < X \leq 1$, $0 < Y \leq 1$). According to the discussion in Sections 3.2.2 and 3.2.3, $Y$ is less than $X$. Then, we can derive formula (8-7) below.

$$\text{DIFF}_{total} = \sum_{j=1}^{N} C * X * A_j + \sum_{i=1}^{M} C * (X - Y) * B_i$$

We consider the most complicated case where upper bound constraints are nested one after another. We suppose that the first upper bound constraint covers $P$ activities. We assume that the distance between any two adjacent upper bound constraints is the same, denoted as $Q$. Since formula (8-7) has nothing to do with SI, we assume that there is no SI. Then, we can derive: $A_j = [P + (M-1) * Q] + j * Q$ and $B_i = P + (i-1) * Q$. If we apply them to formula (8-7), then we have formula (8-8) below.
\[ DIFF_{total} = C \times X \times N \times [P + Q \times (M - 1 + \frac{(N + 1)}{2})] + C \times (X - Y) \times M \times [P + Q \times \frac{(M - 1)}{2}] \] (8-8)

We take a set of specific values to see how formula (8-8) performs. We suppose that \( P = 8, Q = 3, X = 1/2, Y = 1/3, \) and \( C \) is equal to 1 cost unit. We also suppose that \( N \) can change from 0 to 20 and \( M \) can change from 0 to 10. The selection of these specific values is rather random and does not affect our analysis because what we want to see is the trend of how \( DIFF_{total} \) changes with mean WC number \( N \) or mean WI number \( M \). Considering \( M = 0, 2, 4, 6, 8, 10 \), with \( N \) changing, we depict corresponding \( DIFF_{total} \) in Figure 8.1.

![Figure 8.1 Change trend of handling cost difference between conventional work based on two states and our research based on four states by mean WC number](image-url)
(i.e. $N$) and mean WI number (i.e. $M$)

From Figure 8.1, we can see the following facts:

- $M=0$ and $N=0$, then $DIFF_{total} = 0$. In fact, if $M=0$ and $N=0$, there is no any WC or WI. Then, according to Sections 3.2.2 and 3.2.3, the corresponding handling conducted by our research based on SC, WC, WI and SI is the same as that conducted by the conventional work based on CC and CI. Hence, $DIFF_{total}$ is 0.

- Given fixed $M$, with mean WC number $N$ increasing, $DIFF_{total}$ also increases. This means that if there is more WC, based on SC, WC, WI and SI, we can save more handling cost.

- Given fixed $N$, with mean WI number $M$ increasing, $DIFF_{total}$ also increases too. This means that if there is more WI, based on SC, WC, WI and SI, we can save more handling cost.

In summary, only if $N = 0$ and $M = 0$, i.e. there are no any WC or WI, the handling cost of the conventional work based on the two states of CC and CI is the same as that of our research based on our four states of SC, WC, WI and SI. For all other cases, based on SC, WC, WI and SI, we can save the handling cost significantly. Therefore, in overall terms, statistically our four states SC, WC, WI and SI can achieve better handling cost effectiveness significantly than the conventional two states of CC and CI.

We may note that we use the difference between handling cost based on CC & CI and that based on SC, WC, WI & SI to reason about the quantitative analysis. We did not compare the two types of handling cost directly. This is because the handling cost is dependent on two variables $N$ and $M$. If we compare the two types of handling cost directly, we need to split each curve in Figure 8.1 into two curves. The two curves form a pair. Then, there would be 12 curves. However, one curve of a pair will be compared to the other corresponding curves in other pairs. Namely, the
12 curves are not directly comparable with each other. This would be very confusing and would make Figure 8.1 too complicated to be readable. Therefore, we choose to use the difference between the two types of handling cost to conduct the quantitative analysis. On one hand, the quantitative results have been satisfactorily achieved. On the other hand, Figure 8.1 is much readable and easy to understand.

8.3 Summary

This chapter is an extension to this thesis. The objective is to provide a glimpse of the handling of SC, WC, WI and SI, and give an overall picture about the advantage of introducing SC, WC, WI and SI in terms of better temporal verification effectiveness. By this, we can have a more comprehensive understanding about the research in this thesis. We have taken upper bound constraints as the scenario to achieve the objective. Corresponding results, according to Chapter 7, can be equally applied to lower bound constraints and can be simplified and improved for fixed-time constraints.

Specifically, to provide a glimpse about the handling of SC, WC, WI and SI, we have taken WC as an example and developed a method which can utilise mean activity time redundancy of succeeding activities to adjust WC so that statistically it can be kept as SC. That is to say, we need not trigger any exception handling to handle WC. According to Section 3.2.3, we need not do anything for SC. WI can be handled by simper exception handling which is left as future work. SI can be handled by more costly exception handling which is the same as that deployed in the conventional temporal verification work for handling CI.

To present an overall picture about the advantage of introducing SC, WC, WI and SI, we have conducted a quantitative evaluation. The evaluation has demonstrated that we can achieve better handling cost effectiveness significantly with SC, WC, WI and SI than with CC and CI. This, according to Section 2.1, would eventually contribute to better temporal verification effectiveness.
Chapter 9
Conclusions and Future Work

9.1 Summary of this thesis

The research objective described in this thesis is to investigate a series of new concepts, innovative methods and algorithms for temporal verification in grid workflow systems so that we can significantly improve overall temporal verification effectiveness and efficiency. The thesis was organised as follows:

- Chapter 1 introduced temporal constraints and their verification in grid workflow systems. It stated that the types of temporal constraints mainly include upper bound, lower bound and fixed-time. Chapter 1 also described the aims of this work, the key issues addressed in this thesis and the structure of this thesis.

- Chapter 2 overviewed the related work and analysed the research problems in temporal consistency states, one or a few user-defined coarse-grained temporal constraints, checkpoint selection and temporal dependency in terms of temporal verification effectiveness and efficiency. Based on the problem analysis, it was argued that multiple temporal consistency states and a series of fine-grained temporal constraints are needed. Besides, new checkpoint selection strategies which can select only sufficient and necessary checkpoints need to be developed. Temporal dependency between temporal constraints needs to be investigated and incorporated into the verification of
temporal constraints.

- Chapter 3 presented a generic timed grid workflow representation based on the directed network graph (DNG) concept for reasoning about our research. Some time attributes such as maximum or minimum activity duration were denoted. Besides, four temporal consistency states were identified and defined. They were SC (Strong Consistency), WC (Weak Consistency), WI (Weak Inconsistency) and SI (Strong Inconsistency) where CC (Conventional Consistency) corresponds to SC and CI (Conventional Inconsistency) is divided into WC, WI and SI. Based on them, it was briefly analysed that with the four states, better temporal verification effectiveness could be achieved while temporal verification efficiency remained the same as that in the conventional verification work.

Chapter 3 also stated that upper bound constraints were taken as the scenario for reasoning about our research. Then, in Chapter 7, corresponding research results were generalised to all types of temporal constraints including lower bound and fixed-time.

- Chapter 4 developed a novel algorithm on how to assign a series of fine-grained upper bound constraints within one or a few user-defined coarse-grained ones. The detailed assigning process was also provided. The comparison and quantitative evaluation demonstrated that with the assigned fined-grained upper bound constraints, significant improvement on temporal verification effectiveness could be achieved. It was also noted that temporal verification efficiency was about the same as that in the conventional verification work because extra computation by the assigning process was negligible.

- Chapter 5 presented an innovative checkpoint selection strategy which selects only sufficient and necessary checkpoints. Accordingly, the correspondingly unnecessary and omitted upper bound constraint verification
which would otherwise be caused by the conventional verification work can be avoided. The comparison and quantitative evaluation showed that with the strategy, temporal verification effectiveness and efficiency were significantly improved.

- Chapter 6 discussed temporal dependency between upper bound constraints and investigated its impact on temporal verification effectiveness and efficiency. Based on this, new temporal verification algorithms were developed. The comparison and quantitative evaluation showed that with temporal dependency, we significantly improved temporal verification effectiveness and efficiency further.

- Chapter 7 investigated how to apply the research results achieved for upper bound constraints to lower bound and fixed-time constraints. It was shown that the research results from upper bound constraints could be equally applied to lower bound constraints and could be simplified and improved for fixed-time constraints. By this chapter, the research in this thesis has been generalised for all types of temporal constraints.

- Chapter 8 provided an extension to the research in this thesis about the handling of temporal verification results. A WC adjustment method was presented. The quantitative analysis of the improvement by the four temporal consistency states on temporal verification effectiveness was conducted to further support the corresponding conclusion in Chapter 3.

In summary, wrapping up all chapters, we can conclude that with the research results in this thesis, i.e. a set of new concepts, innovative methods and algorithms, we can significantly improve overall temporal verification effectiveness and efficiency.
9.2 Key contributions of this thesis

The significance of this research is that it addresses some of the limitations in the conventional temporal verification work in terms of temporal verification effectiveness and efficiency. The detailed analysis is conducted for each step of the temporal verification cycle, i.e. Step 1 – defining temporal consistency; Step 2 – assigning fine-grained temporal constraints; Step 3 – selecting appropriate checkpoints; and Step 4 – verifying temporal constraints. Based on the analysis, a set of new concepts such as four temporal consistency states, innovative methods and algorithms for temporal verification have been proposed or developed. Corresponding comparisons and quantitative evaluations have shown that these new concepts, innovative methods and algorithms can significantly improve overall temporal verification effectiveness and efficiency. In the context of grid economy, any resources consumed must be paid. Improving temporal verification effectiveness and efficiency would save more resources from temporal verification for executing grid workflows. Therefore, the research in this thesis would eventually improve overall performance and usability of grid workflow systems as temporal verification can be viewed as a function or service of grid workflow systems. As a consequence, by deploying our new concepts and innovative methods and algorithms, grid workflow systems would be able to better support large-scale sophisticated scientific and business processes in the context of grid economy.

In particular, the major contributions of this thesis are:

- **Detailed analysis of the major limitations in conventional temporal verification work in terms of temporal verification effectiveness and efficiency**

  At each step of the temporal verification cycle, this thesis has analysed the major problems in the conventional temporal verification work. At Step 1, this thesis has shown that two temporal consistency states are too restrictive in terms of temporal verification effectiveness. At Step 2, this thesis has analysed the requirements for a number of temporal constraints at various
activities. At Step 3, this thesis has identified that existing representative checkpoint selection strategies often ignore some necessary checkpoints and/or select some unnecessary ones. At Step 4, this thesis has found that the conventional verification work ignores temporal dependency between temporal constraints.

- **Four new temporal consistency states**
  Based on the problem analysis for Step 1 of the temporal verification cycle, four new temporal consistency states have been proposed. They are SC (Strong Consistency), WC (Weak Consistency), WI (Weak Inconsistency) and SI (Strong Inconsistency) where CC (Conventional Consistency) corresponds to SC and CI (Conventional Inconsistency) is divided into WC, WI and SI. Corresponding quantitative analysis has shown that with them, we can achieve much better temporal verification effectiveness.

- **A novel algorithm for assigning fine-grained temporal constraints within one or a few user-defined coarse-grained temporal constraints.**
  This thesis has developed a novel algorithm by which we can assign a series of fine-grained temporal constraints within one or a few user-defined coarse-grained temporal constraints. The corresponding assigning process has been detailed while the extra computation caused by the assigning process is negligible.

- **An innovative checkpoint selection strategy which can only select sufficient and necessary checkpoints**
  To overcome the limitations of existing representative checkpoint selection strategies, this thesis has developed a new strategy. This strategy can select only sufficient and necessary checkpoints dynamically along grid workflow execution. Corresponding selection algorithms have also been designed while the extra computation caused is negligible. The quantitative evaluation has demonstrated that the new strategy can significantly improve temporal verification effectiveness and efficiency.
- **In-depth investigation of temporal dependency and its impact on temporal verification effectiveness and efficiency**

This thesis has investigated the temporal dependency between temporal constraints. It has been found that a series of temporal constraints are often dependent on each other in terms of temporal verification effectiveness and efficiency. The reason for this has also been analysed. That is generally because later temporal verification may make previous temporal verification ineffectiveness and later temporal verification may utilise previous temporal verification to reduce current verification computation for better efficiency. Corresponding new temporal verification algorithms which take the temporal dependency into consideration have been developed while the extra computation caused is negligible. The quantitative evaluation has demonstrated that with temporal dependency, we can significantly improve temporal verification effectiveness and efficiency.

A minor contribution of this thesis is that it has provided an extension about the handling of temporal verification results. The extension would be able to help achieve a more comprehensive understanding of the research in this thesis because it presents some hints about what we will do after we complete temporal verification by deploying the new concepts, innovative methods and algorithms developed in this thesis. In particular, a method has been developed on how to adjust WC so that statistically it could be kept as SC without triggering any exception handling while in the conventional verification work, it would be treated by corresponding exception handling.

### 9.3 Future work

In the future, further investigation into temporal verification related issues in grid workflow systems will be carried out. Future research includes handling WI and SI, integration of theoretical research results in this thesis into existing popular grid workflow systems, new timed grid workflow models, real-world applications and so
In this thesis, four temporal consistency states have been proposed, i.e. SC, WC, WI and SI. For SC, nothing needs to be done. For WC, a method has been developed to adjust it. However, no approaches have been presented to handle WI and SI since they are not within the temporal verification scope. In the future, the approaches for handling WI and SI will be studied and developed. This may include some dynamic negotiations between different grid services in order to compensate corresponding time deficits.

At the moment, the research results in this thesis are general and not specific to any particular grid workflow systems. Hence, another future work is to investigate how to integrate those results into some existing popular grid workflow systems in order to improve their overall performance and usability.

This thesis has introduced some new time attributes including the new four temporal consistency states. Naturally, an interesting topic is whether new timed grid workflow models can be derived, say based on Timed Petri-Nets. With such models, corresponding denotations in this thesis will be made more generic, which would further establish a solid basis for widely deploying the research results in supporting various scientific or business processes.

Finally, for the moment, the research in this thesis is theoretical oriented. We are currently attempting to test the results on some existing grid workflow systems. In future, applications will be further investigated so that practical benefits can be achieved. To do so, collaborative research with other real-world entities needs to be identified and organised.
Bibliography


## Appendix A

### Notation Index

<table>
<thead>
<tr>
<th>Symbols</th>
<th>Denotations</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_i$</td>
<td>The $i^{th}$ activity of a grid workflow</td>
</tr>
<tr>
<td>$C(gw)$</td>
<td>Build-time expected time from which the specification of grid workflow $gw$ will come into effect</td>
</tr>
<tr>
<td>CC</td>
<td>Conventional Consistency</td>
</tr>
<tr>
<td>CI</td>
<td>Conventional Inconsistency</td>
</tr>
<tr>
<td>$CDA(a_p)$</td>
<td>Checkpoint Decision-making Approach at $a_p$</td>
</tr>
<tr>
<td>$CSS_1 \ldots CSS_7$</td>
<td>Previous 7 representative checkpoint selection strategies</td>
</tr>
<tr>
<td>$CSS_{MTR}$</td>
<td>Minimum Time Redundancy based Checkpoint Selection Strategy</td>
</tr>
<tr>
<td>$D(a_i)$</td>
<td>Maximum duration of $a_i$</td>
</tr>
<tr>
<td>$d(a_i)$</td>
<td>Minimum duration of $a_i$</td>
</tr>
<tr>
<td>$D(a_i, a_j)$</td>
<td>Maximum duration between $a_i$ and $a_j$</td>
</tr>
<tr>
<td>$d(a_i, a_j)$</td>
<td>Minimum duration between $a_i$ and $a_j$</td>
</tr>
<tr>
<td>DOMTR</td>
<td>Dynamic Obtaining of Minimum Time Redundancy</td>
</tr>
<tr>
<td>$E(a_i)$</td>
<td>Run-time end time of $a_i$</td>
</tr>
<tr>
<td>$F(a_i)$</td>
<td>Fixed-time constraints at $a_i$</td>
</tr>
<tr>
<td>Symbol</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>$f(a_i)$</td>
<td>Value of $F(a_i)$</td>
</tr>
<tr>
<td>$F_i$</td>
<td>The $i^{th}$ fixed-time constraint when a series of ones exist</td>
</tr>
<tr>
<td>$f(F_i)$</td>
<td>Value of $F_i$</td>
</tr>
<tr>
<td>$L(a_i, a_j)$</td>
<td>Lower bound constraint between $a_i$ and $a_j$</td>
</tr>
<tr>
<td>$l(a_i, a_j)$</td>
<td>Value of $L(a_i, a_j)$</td>
</tr>
<tr>
<td>$M(a_i)$</td>
<td>Mean duration of $a_i$</td>
</tr>
<tr>
<td>$M(a_i, a_j)$</td>
<td>Mean duration between $a_i$ and $a_j$</td>
</tr>
<tr>
<td>$MTR_{SC}(a_p)$</td>
<td>Minimum SC time redundancy at $a_p$</td>
</tr>
<tr>
<td>$MTR_{WC}(a_p)$</td>
<td>Minimum WC time redundancy at $a_p$</td>
</tr>
<tr>
<td>$R(a_i)$</td>
<td>Run-time completion duration of $a_i$</td>
</tr>
<tr>
<td>$R(a_i, a_j)$</td>
<td>Run-time completion duration between $a_i$ and $a_j$</td>
</tr>
<tr>
<td>RTDA</td>
<td>Run-time Time Deficit Allocation (for adjusting WC)</td>
</tr>
<tr>
<td>$S(a_i)$</td>
<td>Run-time start time of $a_i$</td>
</tr>
<tr>
<td>SC</td>
<td>Strong Consistency</td>
</tr>
<tr>
<td>SI</td>
<td>Strong Inconsistency</td>
</tr>
<tr>
<td>$U(a_i, a_j)$</td>
<td>Upper bound constraint between $a_i$ and $a_j$</td>
</tr>
<tr>
<td>$U_i$</td>
<td>The $i^{th}$ upper bound constraint when a series of ones exist</td>
</tr>
<tr>
<td>$u(a_i, a_j)$</td>
<td>Value of $U(a_i, a_j)$</td>
</tr>
<tr>
<td>$u(U_i)$</td>
<td>Value of $U_i$</td>
</tr>
<tr>
<td>WC</td>
<td>Weak Consistency</td>
</tr>
<tr>
<td>WI</td>
<td>Weak Inconsistency</td>
</tr>
</tbody>
</table>