國 立 交 通 大 學

資 訊 工 程 學 系

碩 士 論 文

工作流程規格資源分配限制的遞增式分析

Incremental Analysis of Resource Constraints for Workflow Specifications

研究生：楊大立

指導教授：王豐堅 教授

中華民國九十四年二月

工作流程規格資源分配限制的遞增式分析

Incremental Analysis of Resource Constraints for Workflow Specifications

研究生：楊大立　　Student：Daly Yang

指導教授：王豐堅 博士　　Advisor：Dr. Feng-Jian Wang

國立交通大學

資訊工程學系

碩士論文

A Thesis
Submitted to Department of Computer Science and Information Engineering
College of Electrical Engineering and Computer Science
National Chiao Tung University
In Partial Fulfillment of the Requirements
For the Degree of Master
In

Computer Science and Information Engineering

February 2005

HsinChu, Taiwan, Republic of China

中華民國九十四年二月

工作流程規格資源分配限制的遞增式分析

研究生：楊大立　　指導教授：王豐堅 博士

國立交通大學

資訊工程研究所

新竹市大學路 1001 號

摘要

工作流程管理技術提供有效的方法，模組化並控制組織內部或組織與組織間複雜的企業流程。工作流程管理系統大部分都提供兩項主要元件：設計環境與執行環境。流程設計師在設計階段定義工作流程的規格後，必須經過適當的驗證，才能夠於執行階段正確的執行。驗證工作流程規格的方向主要分為結構、時間、與資源三個方面，本文提出一系列的方法，針對工作流程的每次編輯及所造成的影響作時間與資源分配限制的分析，最後以遞增式的方法改善效能，同時將資源衝突的詳細內容資訊提供給流程規格的設計師或維護者。

關鍵字: 工作流程規格、資源分配限制、遞增式分析

Incremental Analysis of Resource Constraints for Workflow Specifications

Student: Daly Yang Advisor: Dr. Feng-Jian Wang

Institute of Computer Science and Information Engineering

National Chiao Tung University

1001 Ta Hsueh Road, Hsinchu, Taiwan, ROC

Abstract

In general, a workflow management system (WfMS) consists of two main functional components: modeling component and enactment component. A workflow designer defines workflow specifications by modeling component in the build-time, and these specifications have to be verified before they can be executed correctly. There are three aspects of workflow verification, which are structural, temporal, and resource verifications. In this thesis, we discuss several approaches for analysis of resource constraints, and an incremental approach of resource consistency with improved performance and detailed information about each resource conflict is represented.

Keywords: workflow specification, resource constraints, incremental analysis

誌謝

本篇論文的完成，首先要感謝我的指導教授王豐堅博士兩年多來不斷的指導與鼓勵，讓我在軟體工程及工作流程的技術上，得到很多豐富的知識與實務經驗。另外，也非常感謝我的畢業口試評審委員葉義雄博士以及留忠賢博士，提供許多寶貴的意見，補足我論文裡不足的部分。

其次，我要感謝實驗室的伙伴們，有博士班嘉麟學長、靜慧學姊、懷中學長督導我寫論文，對論文給予了相當多的寶貴意見，而其餘幾位學長姐熱心地參與幫忙和討論，讓我學得許多論文技巧，得以順利的撰寫論文。當然，值得一提的是我們這屆畢業生吉正、瓊文及祖年，在各方面彼此不斷的砥礪與照顧下，使得大家在各個領域的技術及理論上能有所成長。

最後，我要感謝我的家人及朋友，由於有你們的支持與鼓勵，讓我能心無旁騖地讀書、作研究然後到畢業。由衷地感謝你們大家一路下來陪著我走過這段研究生歲月。

目次

圖表列表

演算法列表

表格

Chapter 1. 簡介

工作流程電子化是一連串的程序規則的自動化，而根據這些規則將有關的文件、資訊和任務由一個參與者傳遞給下一個參與者，以達成特定的目的[1][28]。一般的作業或是工作大多並非單獨一個人可以完成，尤其是在稍具規模的企業組織裡，需要許多工作流程來規範整個企業之各項作業流程。工作流程管理用以自動地協調、控制及溝通企業營運中所需要的工作項目，滿足作業流程的需求，改進工作時間的協調和增加工作的自動化程度，使工作流程更具效率。工作流程管理可應用於公司企業、組織內部，或是整合運用於上下游廠商或同盟間相關的工作流程。面對日趨激烈的競爭，適當的管理可大幅縮短過於冗長不適的工作流程，精簡組織人事的開銷，讓員工的產能大幅提升，達成企業再造的目的。回顧過去十年，工作流程因應網路發展的加速而有一波蓬勃發展 [29]，工作流程管理系統 (WfMS) 的開發是目前資訊系統領域當中一項重要的課題。

現今的工作流程管理系統，皆提供設計階段與執行階段的環境。工作流程設計者或維護者在設計階段進行工作流程規格的規劃、設計，經過驗證、測試後，交由執行環境產生對應之工作流程實例，然後由參與的使用者去完成流程中的個別任務。一般而言，流程管理者則可以在這兩個階段進行分析、評估、控管與監看。

1.1 動機

爲了確保執行階段能夠正確執行設計階段所制定的規格，必須在執行工作流程實例之前做好規格定義的驗證工作。驗證一個工作流程規格的正確性，主要朝向結構 (structural)、時間 (temporal)、與資源 (resource) 這三方面，其中

結構檢驗較優先於其他兩項，因爲當一個工作流程規格定義結構不正確時，可能連執行都無法啓動。 Figure 1 [19]顯示一個工作流程規格定義的生命週期。此生命週期當中首先是分析企業流程需求，依據所使用的工作流程管理系統特定的描述語言定義其規格，待定義完成後進行結構上的檢查，結構上的檢查通過後進行時間、資料、資源等相關的檢驗(verification)。由設計者根據檢驗的結果進行測試(validation)，同時可以根據模擬的結果進行最佳化工作(optimization)。最後階段是部署(deployment)、執行與維護的階段，流程規格可能依執行效率、或企業需求變動，進行規格修改，然後依照上述驗證、測試與部署動作再重覆執行。

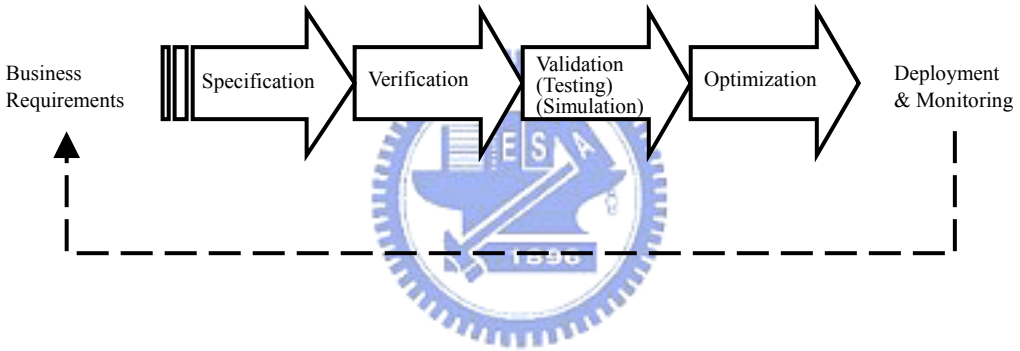


Figure 1 Life Cycle of Workflow Specification [19]

綜觀早期整個電子化流程開發程序的各個階段，由於缺乏有效的連結監控機制，導致開發者必須以人工方式檢視電子化流程的正確性，或是花費大量的人力和資源來計算、維護電子化流程設計的結果，使得整個流程開發成本居高不下。在工作流程定義階段，人工定義所有工作流程所需的活動 (activity)、文件 (artifact)、資源 (resource) 和限制(constraint) 等等。這樣繁複的定義，難免因爲人爲疏失導致錯誤產生，在現有的系統當中，少有即時的給予警訊通知，告知設計者這樣的行爲可能發生的錯誤，以及錯誤產生點及錯誤所波及的元件。所以由 Figure1 我們可以明確的知道，這些在第一階段所發生的錯誤，可

能引起的漣漪效應，如果不儘早解決的話，對於系統可能會造成很大的困擾，因爲錯誤發展到最後所涵蓋的範圍，可能會超出我們可以控制的範圍，最糟的情況有可能讓錯誤無限擴張。

工作流程規格定義在結構與時間驗證方面已有許多有效的方法、文獻[3][6][7][10][13][14] 已被提出，但目前探討資源分配限制的驗證，僅有靜態的全面性(totally)檢驗，這樣的檢驗當遇到流程中 processes 數很多時，一方面檢驗缺乏效率，另一方面不容易找到錯誤的源頭。工作流程規格的設計師或維護者，在設計階段需要即時且充足的相關資訊來輔助其進行各種流程規格的編輯，比方說，新增 process 產生哪些相關的資源衝突、或刪除 process 消去哪些相關資源衝突，如此可避免錯誤擴大與複雜化。對每次編輯進行局部適當、適時、適量的檢驗，而非全面性耗時的批次檢驗。在假設工作流程規格結構正確的前提下，本論文研究重心擺在遞增式分析流程規格中資源衝突的檢驗。Figure 2 即爲運用遞增式分析工作流程規格的設計，其中即時的檢驗能將編輯後的相關資訊，提供流程的設計師或維護者，藉以提升工作流程設計、維護的效率。

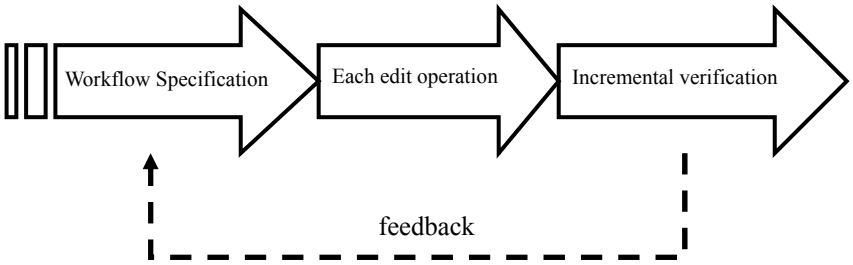


Figure 2 Incremental life cycle of editing a workflow specification

1.2 方法與步驟

本論文首先探討現有工作流程規格的正規模式 (formal model)，並說明採用 DAG (directed acyclic graph) 模型做爲分析的基礎架構之理由。根據此基礎架構，針對使用者對於工作流程規格可能的編輯動作做分類，然後依據這些不同的編輯動作，採取一系列互動式與遞增式檢驗的探討與分析。

考慮時間因素下，當工作流程規格中各個 process 有充分的相關時間資訊時，資源衝突的判斷可以更加精確，此外會因此衍生新的編輯動作，例如改變工作流程的工作時限 (durations) ，因應這些編輯動作，必須額外增加分析的內容。同樣的在考慮時間因素的情況下，探討可行的互動式與遞增式的檢驗演算法。

1.3 目標

現有的資源衝突檢驗技術 [2] 包含以下這些缺點：

(1) 批次檢驗的過程過於冗長，效率不佳。

(2) 針對是否產生資源衝突變化的檢驗，無法迅速達成。

(3) 未能詳細列出發生衝突的 processes 間，及其所衝突的相關資源。

(4) 未能從設計流程規格的使用者觀點作檢驗分析。

本論文提出的方法目的是希望能快速提供給使用者是否產生、消除資源衝突的資訊外，亦能提供各項資源衝突的細節資訊。此外，透過互動式與遞增式不同的演算法探討以分析如何提升資源衝突檢驗效率。

1.4 各章節簡介

本論文其餘部分的構成如下：第二章描述工作流程管理、工作流程規格與現有的資源衝突檢驗技術。第三章介紹如何針對流程規格的編輯動作進行互動式與遞增式資源衝突檢驗。第四章探討加入時間因素後，分析互動式與遞增式資源衝突檢驗。第五章討論時間複雜度，並以一個劇本範例來進行說明。第六章的部分爲相關研究。第七章針對本論文的貢獻與未來的研究方向做總結。

Chapter 2. 背景說明

2.1 工作流程管理技術

1996 年，美國國防部支持工作流程的初始研究，開啓了 WfMC [1] 在這個領域的研究。最顯著的研究成果是由五個功能介面描述的工作流程參考模型 (Workflow Reference Model)。這個組織組成小組撰寫這些介面的定義。Figure 3 即描述 WfMC 所提出的工作流程參考模型。[9]

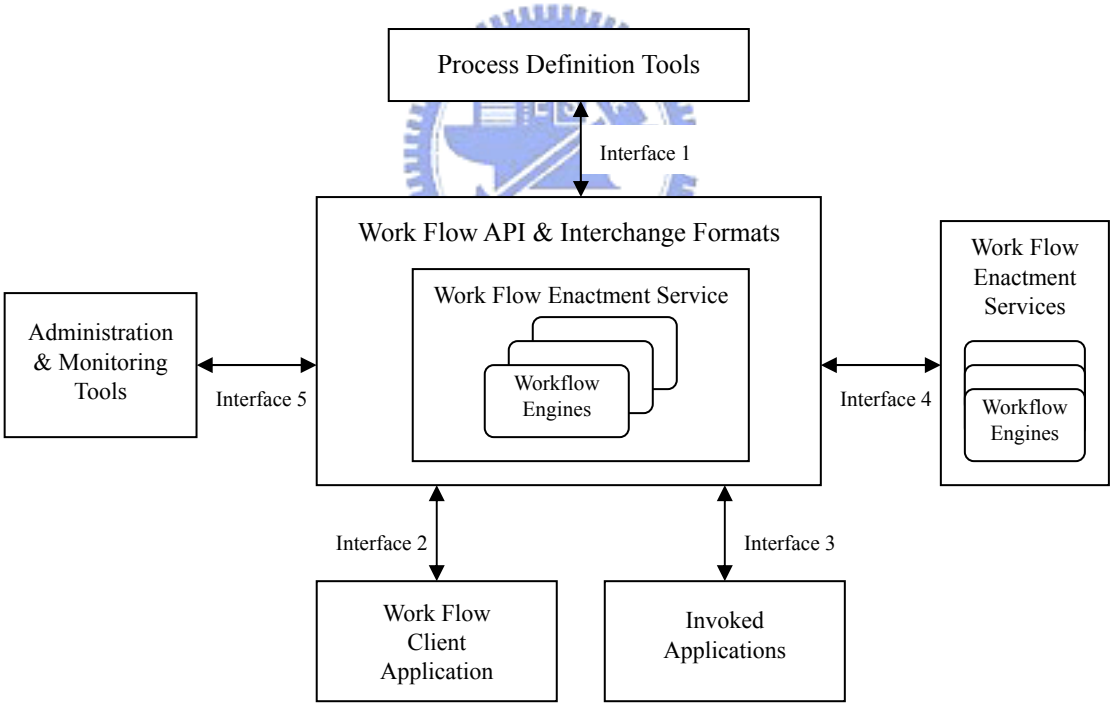


Figure 3 Workflow Reference Model Components and Interfaces

根據工作流程管理組織 (WfMC) [1] 的定義，工作流程爲一連串完整、或局部的自動化企業程序，根據一組規則，將活動由一個參與者，利用文件、資訊與任務項目的方式傳遞給另一個參與者，以達成企業目標。工作流程管理系統定義、組成並管理工作流程，使其透過軟體的使用以及一個或數個的工作流程引擎來執行，且這些工作流程會根據流程規格的定義，與流程參與者產生互動，必要的話也可以呼叫資訊技術的工具及應用程式來使用。

一個工作流程管理系統根據預先定義好的商業程序定義，決定工作的流程。這樣的系統管理達成目標所需要的資源(包括了應用程式、資料、以及人物)，同時提供監看的便利性與控制的能力。一般而說，這樣的系統常被用於大幅減少停滯的時間、搜尋相關活動，以及供給線的傳輸延遲。[8]

2.2 描述工作流程規格的正規模式

目前以正規模式 (formal model) 描述工作流程的模式常見的有：

(1) Petri-nets [6][10][24]: Petri-nets 可以表達較複雜的工作流程關係，Colored Petri-nets [25][26]多加了 multiple token 及 color 的觀念，用以詮釋更複雜的狀況。功能雖強大，但是複雜度過高，僅適合當作執行模式，不適合直接用於描述工作流程。

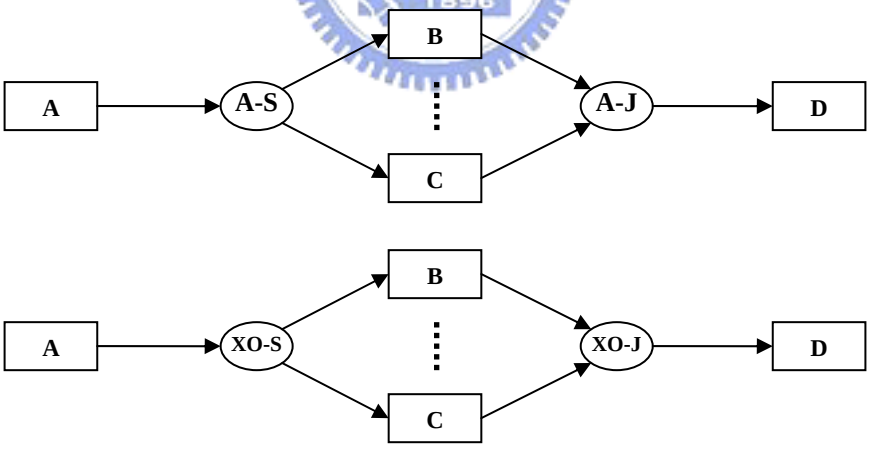
(2) 狀態圖 [12]: 狀態圖的方法主要用於描述系統之活動間的控制流並驗證是否可到達最後狀態 (Reachability)。

(3) DAG [3][11]:DAG (directed acyclic graph) 的優點在於方便驗證資料流與控制流的一致性，比較符合我們設計階段流程規格分析的需要，本論文接下來討論到的工作流程規格，將採用 DAG 的描述方式。

2.3 基本假設與符號定義

工作流程規格描述一個或數個工作流程實例 (instances) 的正規定義，並記錄參與的 processes、流向、資料，以及資源。正規的工作流程規格定義，可以用 3-tuple **< N, F, R>** 方式來表示一個工作流程規格 ws，其中 **N** 表示參與的 processes 集合 $\{n_1, n_2, ..., n_k\}$，**F** 定義流向集合$\{f_1, f_2, ..., f_m\}$，**R** 記錄各 process 操作時會使用、參考到的相關資源集合之總集合。

上述 processes 集合$\{n_1, n_2, ..., n_k\}$當中的每個 process 可區分爲兩種類別，一種是 activity processes，而另一種是 control processes。activity processes 負責執行任務，常見的執行方式是透過人機互動、或是以軟體代理程式 (agent) 自動完成。control processes 屬於邏輯式的選擇，目的是主動根據策略作出決策。本文中討論四種類型的 control processes，分別是：(1) and-join, (2) and-split, (3) xor-join, (4) xor-split, 這些點的說明與解釋詳細列於 Figure 4 當中。



A, B, C, D: Activity processes that perform tasks
A-S: And-Split that forks into each successor activity with related data
A-J: And-Join that synchronizes each parent activity with related data
XO-S: Exclusive Or-Split that chooses one successor activity to perform
XO-J: Exclusive Or-Join that merges one parent activity to one activity

Figure 4 Control processes

一個工作流程規格，擁有唯一的起始點，與數個結束點。**F** 定義工作流程中流向集合{ $f_1, f_2, ..., f_m$ }，每一條流向 f_i 以 $f_i =< n_a, n_b >$ 的方式來記錄，描述 process n_a 執行後執行 process n_b ，其中 $n_a, n_b \in \mathbf{N}$ 。

資源集合 **R** 描述表示爲{R_1, R_2, …, R_n }記錄工作流程中各 activity process 所需要的資源，每一個 R_i set 分別對應到 activity process n_i ，set 當中記錄相關所需的哪些資源 r_i 集合。

Definition 1 (Workflow Specification)

ws = <N, F, R> is a workflow specification where

(1) N: a set of processes

$\forall n \in \mathbf{N}$, n **.TYPE** $\in$ **{ACTIVITY, AND-SPLIT, XOR-SPLIT, AND-JOIN, XOR-JOIN}**

(2) F: a set of flows

$\forall f \in \mathbf{F}$, $f =< n_i, n_j >$, **there exists a transition from** n_i **to** n_j

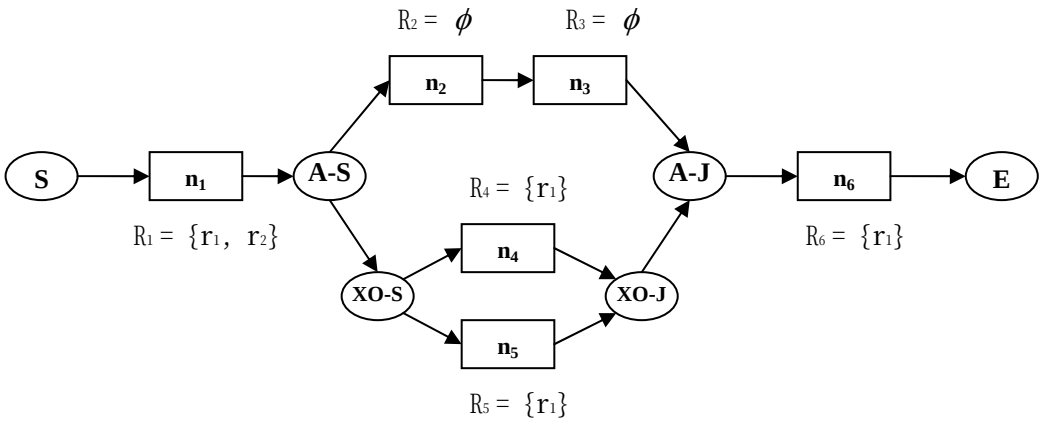
f **is an out-flow of** n_i, **and** f **is an in-flow of** n_j **respectively**

n_i **is a source process of** f, **and** n_j **is a sink process of** f **respectively**

(3) R: a set of resource sets

$\forall R_i \in \mathbf{R}$, R_i **= {** r_j **|** r_j **is a resource accessed by** n_i, $n_i \in \mathbf{N}$ **}**

底下 Figure 5 舉例說明，以一個有向無循環的圖形(DAG)，可大致表示一個工作流程規格所使用的正規定義。每個資料集合 Ri 當中的元素表示此所需參考、使用到的資源集合，ϕ 則表示此 activity process 不需要參考或使用任何資源。



n_i : An activity process of **N** that perform task(s)

S: Start node

E: End node

R_i **:** a set of resource accessed by process n_i

r_i **:** a resource

Figure 5 A basic workflow specification diagraph

接著定義路徑(path)、可達性(Reachability)、路徑長度(distance)、以及最近共同祖先(nearest common ancestor)等項目，這些項目將會在後續的討論當中使用到。[2]

Definition 2 (Path and acyclic path)

Let $p = < n_1, n_2, ..., n_t >$**, where** $n_i \in N, i = 0,1,...,t$**, be a sequence. If** $< n_i, n_{i+1} > \in$ **F, where** $n_i, n_{i+1} \in N, i = 0,1,...,t-1$**, then** p **is called a path on ws. The length of** p **is** t**, denoted as** $|p| = t$**.**

If $n_i \neq n_j$ **for** $i \neq j$**, where** $i,j=0,...,t$**, then** p **is regarded as an acyclic path on ws.**

Definition 3 (Reachability)

Process n_j **is reachable from process** n_i **if there is an acyclic path** $p =< n_i,...,n_j >$ **on ws.**

Let $\text{Reachable}(n_i, n_j)$ **be a Boolean function to denote the reachability from process** n_i **to** n_j **such that**

$$\text{Reachable}(n_i, n_j) = \begin{cases} \text{TRUE, if } (\exists\ p =< n_i,...,n_j >), \\ \text{FALSE, otherwise.} \end{cases}$$

Definition 4 (Distance)

The distance between two processes n_j **and** n_i **in a workflow specification can be computed as follows:**

$$\text{Distance}(n_i, n_j) = \begin{cases} \text{MIN}\{|\ p_s\ ||\ p_s =< n_i,...,n_j >, s = 1,...,m\}, \text{if (Reahable}(n_i, n_j) = \text{TRUE}) \\ \text{MIN}\{|\ p_s\ ||\ p_s =< n_j,...,n_i >, s = 1,...,m\}, \text{if (Reachable}(n_j, n_i) = \text{TRUE}) \\ +\infty, \text{otherwise.} \end{cases}$$

Definition 5 (Nearest common ancestor)

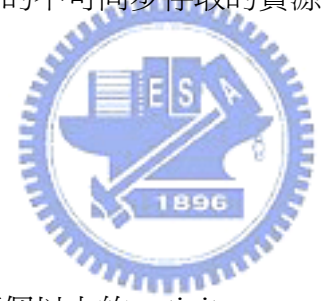
Given two processes n_i **and** n_j **in a workflow specification, their nearest common ancestor is the process, which is a common ancestor and has the shortest distances to them, denoted as** n_{nca} **.**

2.4 資源限制的一致性

2.4.1 資源

在一個工作流程中，activity process 常常需要存取部分資源以協助其完成預期的工作內容。資源包含軟體、硬體設備，例如印表機、文件資料，亦或是中央資料庫中的一筆記錄等等。

資源大致上可分爲共享資源或是私有資源，由於私有資源屬於該活動點自我運作時參照使用，所以不需列入資源一致性分析。共享資源當中，允許多個活動同時存取、或是依據活動存取模式決定資源可否被存取的類型，也不列入考量。就是僅考量屬於嚴格的不可同步存取的資源。[2]



2.4.2 潛在的資源衝突

資源衝突是指兩個或兩個以上的 activity processes，在執行時期同時間參考到共同的資源。在同一工作流程規格中，兩個 activity processes 會產生潛在資源衝突 (potential resource conflict)，須符合下列兩項檢驗條件：

(1) activity processes $n_i, n_j \in N$ 且 $i \neq j$，若 n_i, n_j 所需要存取的資源有交集，亦即 $R_i \cap R_j \neq \phi$，則稱 n_i, n_j 有「資源相依」關係。(resource dependency)

(2) n_i, n_j 不在同一條路徑上，且其最近共通祖先點 (nearest common ancestor) 有 AND-SPLIT 類型的控制點。也就是說，n_i, n_j 處在可能同時執行的兩條平行路徑上。

以上是 Hongchen Li [2] 採用結構的分析，對資源一致性的檢測。這些檢測工作，是在整個流程規格規劃完畢後，才進行檢測，而檢測方式也是依序以每一個 process 作出發，去檢查整個流程架構中其他 process，先判斷是否有用到相同資源的，接著比較彼此間是否互相可以到達 (reachable)，所謂可以到達的意思，就是處在同一條路徑 (path) 上，因爲同一路徑上的 activity processes，意味著執行的先後順序，可以確保不會同時執行，這樣便不用繼續檢查。若是檢查到的此兩 processes 未符合上述條件，而且有資源相依的關係，那麼就要接著判斷他們的最近共通祖先點 (nearest common ancestor)，然後藉由判斷此 control process 的特性，若是 XOR-SPLIT 則表示兩個 processes 的執行爲互斥、執行時期不會同時進行的兩條路徑上，但若是 AND-SPLIT 就可能形成潛在的資源衝突。

2.4.3 加入時間因素考量的資源衝突

如上一小節所提到，資源相依的兩個 processes 處在平行且其最近共通祖先爲 AND-SPLIT 的情況下，才可能導致潛在的資源衝突。然而即使同時執行的平行路徑上具資源相依的兩個 processes，也未必會形成資源衝突，原因是彼此的工作時間區段未必會有交集。

Marjanovic [4] 針對時間分析定義了 activity process 的最小工作時限 (duration) 與最大工作時限的觀念，藉由這兩項定義，我們可以推導出各相關 process 的最早開始工作時間 (earliest start time) EST 與最晚完成工作時間 (latest end time) LET 值。Figure 6 介紹這些相關的計算方式。其中 d(A) 與 D(A) 分別表示 process A 的最小工作時限值與最大工作時限值。

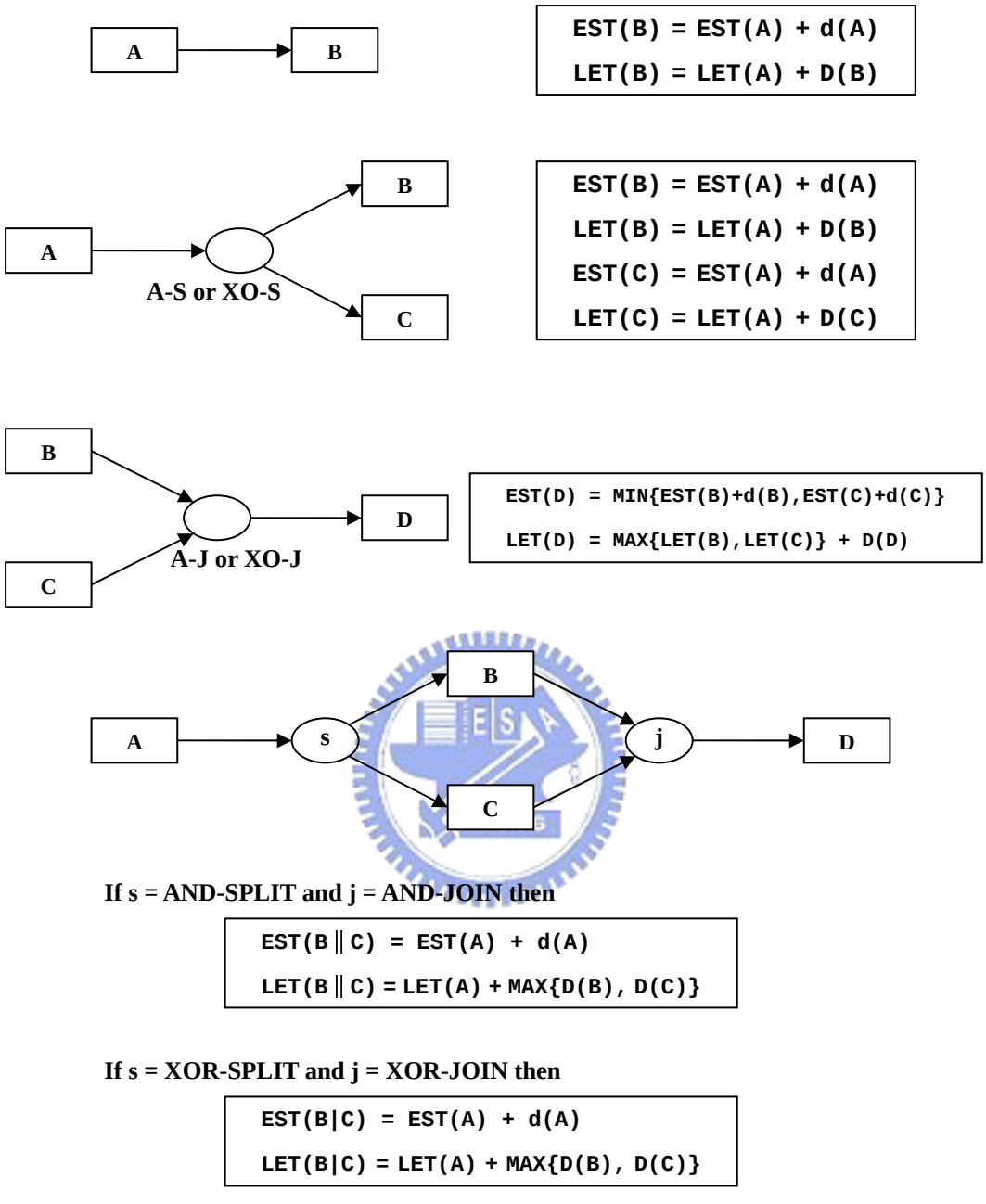


Figure 6 Estimated active interval

因此若是工作流程規格的定義，能夠記錄各 activity processes 的最小與最大工作時限，便可以利用這些記錄，去推算各 activity process 的 EST 與 LET 值，協助我們對資源衝突驗證加入時間的考量，而上一小節提到的潛在的資源衝突成立條件就須額外加入下列項目：

(3) n_i, n_j 各別預估的工作時間區段，有所交集。即 $[\text{EST}(n_i), \text{LET}(n_i)] \cap [\text{EST}(n_j), \text{LET}(n_j)] \neq \phi$。其中 $[\text{EST}(n_i), \text{LET}(n_i)]$ 意味著點 n_i 最大的可能工作時間區段 (EAI, Estimated active interval)。

2.5 基本功能函數與批次的資源衝突檢驗演算法

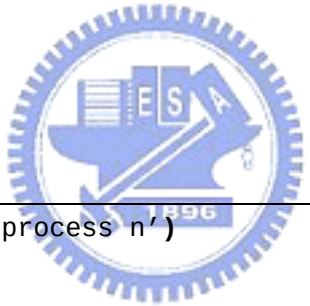
給定 2.3 節當中 Definition 3 (Reachability) 與 Definition 5 (Nearest common ancestor)，則可撰寫出底下兩個便利的基本功能函數：

可達性函數 Reachable(n, n') 檢查是否存在一條路徑可由 process n 到達 process n', 若是存在則回傳 TRUE，否則回傳 FALSE。當回傳爲 TRUE 時意味著 n 爲 n' 之 ancestor process，由於隱含執行的先後順序，意即 n 確定在 n' 前執行，確保了彼此間不會有資源衝突情況。可達性函數具有映射性 (reflexive)，也就是 Reachable(n, n) 會回傳 TRUE。

NCA(n, n') 負責傳回 process n 與 process n' 的最近共同祖先點 (nearest common ancestor)，而此點對於下一節提到的檢測演算法非常有幫助。[2][3] 中也證明了在一個結構正確的工作流程規格中，任兩個 processes 的最近共通祖先點，其實都會屬於同一種 control process，亦即都是 AND-SPLIT 或是 XOR-SPLIT。

```
Boolean Reachable(process n, process n')
1 {
2  if n = n' then return TRUE;
3  insert all out-flows of n into an empty Queue Q;
4  while Q ≠ ϕ
5  {
6      remove the first flow f from Q;
7      n_i = sink process of f
8      if n_i = n' then return TRUE;
9      insert out-flows of n_i into Q;
10  }
11  return FALSE;
12}
```

Algorithm 1 Reachable Algorithm



```
process NCA(process n, process n')
1 {
2  if Reachable(n, n') then return n;
3  if Reachable(n', n) then return n';
4  insert all in-flows of n into an empty Queue Q;
5  while Q ≠ ϕ
6  {
7      remove the first flow f from Q;
8      n_i = source process of f
9      if Reachable(n_i, n') then return n_i;
10     insert all in-flows of n_i into Q;
11  }
12}
```

Algorithm 2 Algorithm for nearest common ancestor

```
Boolean Batch_Verification(workflow specification ws)
1 {
2   process set S = { all activity processes in ws };
3   calculate EST(n) and LET(n) for each process n in S;
4   Boolean b = TRUE;  //return FALSE if there exists resource conflicts
5   While S ≠ ϕ
6   {
7     remove a process n from S;
8     for each n_i in S
9     {
10        if R(n_i)∩R(n)=ϕ then   // R(n) returns resource set of n
11            skip to the next iteration;
12        else if Reachable(n_i,n) or Reachable(n_i,n) then
13            skip to the next iteration;
14        else if NCA(n_i,n).TYPE=AND-SPLIT then
15            if [EST(n_i),LET(n_i)]∩[EST(n),LET(n)] ≠ ϕ then
16            {
17                printf ("Potential resource conflict between", n_i,
                  "and", n);
18                b = FALSE;
19            }
20    }
21  }
22  return b;
23 }
```

Algorithm 3 Batch verification of resource conflicts

Algorithm 3 爲 [2] 中對潛在資源衝突以非遞增方式進行全面性檢驗的演算法，此演算法有加入時間因素的考量。首先是將所有的 activity processes 置入一個集合 S，再依序取出一個 process，針對此 process 與其他剩下的 processes 作比對，檢驗其使用的資源是否有交集、彼此是否具可達性。若有資源交集且互不可到達，則再判斷它們的最近共通祖先點 (NCA) 的類別，若爲 AND-SPLIT 且彼此最大的可能工作時間區段 (estimated active interval) 有重疊 (overlap) 現象時，則此演算法顯示有潛在資源衝突。在此處若消去最大的

可能工作時間區段 (EAI) 是否有交集的判斷，則爲不考慮時間因素的潛在資源衝突的檢驗。

整體的時間複雜度，假設整個流程規格的 processes 數爲 n，平均路徑長爲 k，S 的兩層迴圈耗時 (n-1)+(n-2)+…+1 = n(n-1)/2 即 $O(n^2)$，後續三個 if 及相關的 else if 比較式當中，第一個比較資源參考重複性僅需常數時間、第二個比較 Reachable，爲任兩個 processes 間的對應距離，最後的比對是從 NCA 的函式中可明顯看出，process n 以回溯方式往回推至 ancestor process n_i，再檢驗 Reachable(n_i, n')直到成立。考慮下圖 worst case 的情況，NCA 的判斷並不會發生，所耗的時間在比較 Reachable 上爲 O(n)，因此總體的時間複雜度歸納爲 $O(n^3)$。

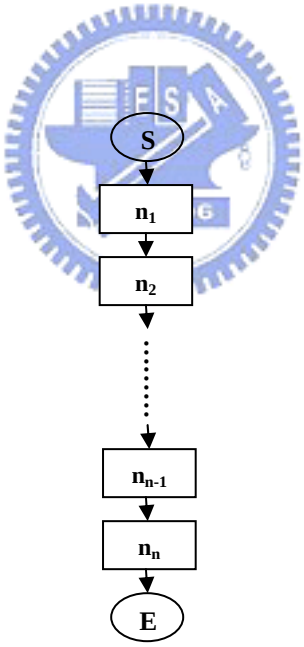


Figure 7 a worst case example of Batch verification

Chapter 3. 資源分配限制分析

本章描述工作流程規格於編輯操作時的資源衝突檢測分析，這些分析演算法主要包括：(1)有無產生資源衝突的檢查，(2)提供資源衝突細節的資訊，(3)利用背景知識加速分析工作。前兩者可稱爲式互動式的資源衝突分析，而第三項則稱遞增式的資源衝突分析。在工作流程設計師或維護者編輯工作流程規格時，以這些演算法提供即時的相關資訊。描述這些檢驗的演算法前，本章第一節當中先以直觀的方式，改寫 **Algorithm 3** Batch verification，做爲互動式演算法的一種方法，並討論其效率。

3.1 批次資源衝突檢驗演算法的改良

觀察 **Algorithm 3** 當中，只要將此演算法除去外層 WHILE 迴圈，即可改爲互動式分析一個 activity process 與其他 processes 間資源衝突的情況。以下 **Algorithm 4** 描述這個改良過的批次資源衝突檢驗演算法，傳入的變數當中增加了 process n，表示只針對 process n 檢查是否與其他的 process 存在資源衝突。

```
Boolean Modified_Batch(workflow specification ws, process n)
1 {
2   process set S = { all activity processes in ws };
3   Boolean b = TRUE;  //return FALSE if there exists resource conflicts
4
5   remove n from S;
6   for each n_i in S
7   {
8     if R(n_i)∩R(n) =ϕ then   // R(n) returns resource set of n
9         skip to the next iteration;
10    else if Reachable(n_i,n) or Reachable(n_i,n) then
```

```
11          skip to the next iteration;
12      else if NCA(n_i,n).TYPE=AND-SPLIT then
13      {
14          printf ("Potential resource conflict between", n_i,
            "and", n);
15          b = FALSE;
16      }
17    }
18    return b;
19 }
```

Algorithm 4 Modified from batch verification

上述演算法的時間複雜度雖會將 Batch 的方法由 $O(n^3)$ 降爲 $O(n^2)$，但採取這樣的檢驗方法仍會消耗過多時間做額外的比較、不適合用於互動式分析。下一節我們介紹較佳的演算法來作較執行與使用者的互動式分析，並提供較細膩的相關資訊。

3.2 activity process 編輯操作與造成的影響

流程設計師在設計階段，對工作流程規格中一個 process 所作的可能操作有：新增、刪除、修改。如 2.2 中所述，process 的類型主要區分爲兩種：(1) activity processes (2) control processes。前者主要提供給使用者任務項目內容，後者主要作爲流程流向的控制。由於 control processes 的修改可能會造成工作流程結構上的破壞，必須另外由結構驗證的方法來加以分析，而這些分析超出本文的範圍，且以下所討論的工作流程規格，皆在無迴圈且結構正確的前替下。因此我們將問題聚焦於針對 activity processes 的編輯作分析。

互動式檢驗分析，目的是針對流程設計師的每一個編輯動作進行資源衝突分析，提供即時反應機制，使設計師能夠迅速的獲知每個編輯細節動作，影響潛在資源衝突的情況。[2] 中陳述的資源檢驗方式雖說完整，但全面性檢查所有點的方法並不符合互動式開發時的分析，也就是說，流程的設計師無法準確的掌握該次編輯異動所造成的影響爲何。除此之外，若以這樣的檢驗資源衝突演算法分析細微修改則會效率不彰。我們期望能找出一個符合互動式設計的檢測方法，能夠讓設計師在設計階段的環境中編輯異動一個 process 時，便能以更有效率的演算法，檢查出這個異動是否會帶來資源衝突的衝擊。

activity process 的編輯主要可以分爲四種，分別是： (1) 新增一個 activity process， (2) 刪除一個 activity process， (3) 增加一個 activity process 參考的一項資源， (4) 刪除一個 activity process 當中已被參考的一項資源內容。稍後分別再對這些情況的操作與影響做更詳細的說明。

從以上四種情況來看，對於被修改的 activity process n 而言，下述三種情況的 processes 都不會與其發生資源衝突的增加或消除，因此在遞增式分析時可以加以省去相關的比對工作。這些可省去比對的 processes 包括有：(1) 所有可到達 process n 的先行(ancestor) processes，(2) 所有 n 可經由 paths 到達的後續(descendant) processes，(3) 與 n 處在不同的平行 path 上，但彼此的共同祖先爲 XOR-Split。

當一個新的 activity process 被插入工作流程當中時，如果此 process 有參考到任何資源，則有必要進行是否會造成資源衝突的檢驗。首先判斷此新增 process 的導入流 (in-flow) 的來源點，也就是此新增點的父 process，判斷是否爲一個 AND-SPLIT 型態的 control process，若不是則由此回溯的方式依序尋找其來源點，直到 Start node 爲止。當找到 AND-SPLIT ancestor process 時，即可抓出所有可能同時執行的平行路徑，然後判斷這些平行路徑上的各 process，是

否與此新增的 activity process 有資源的交集，如果交集不爲空集合，則符合不考慮時間因素的資源衝突條件，也就是可能會導致一種或多種不同資源的潛在資源衝突發生。

當一個 activity process 被刪除掉時，由於其參考到的資源集合也相對的被刪除，因此若此資源集合不爲空集合時，則可能使原先多種資源的潛在資源衝突的情況消除掉。必須在該 process 確實被刪除前，檢查出已存在衝突，才能告知使用者。

若增加 activity process 某項資源的參考，.所需要判斷的 process，即爲與此點所在路徑可能同時執行的平行 (parallel) 路徑上其他 processes。此時需檢驗這些路徑上的 processes 個別所含的資源集合是否包含了此項新增的資源，若有包含則可能發生此單一種資源類別的資源衝突。

修改時若僅是刪除掉所參考的資源，則可能消除同時執行的平行路徑上的資源相依 processes 的單一種資源的資源衝突。類似於刪除一個 process 的分析，在確切消除該資源前，必須進行相關的檢驗，以告知使用者在特定的 process 上刪除該項資源會帶來哪些資源衝突的消除。

3.3 互動式資源衝突檢驗演算法

如同前節中對一個 activity processes 編輯異動的分析，互動式檢驗演算法可改良爲分析異動點所帶來的影響即可，而不必如 **Algorithm 4** 中對整體工作流程規格作完整的比對。底下 **Algorithm 5** 用於判斷新增插入一個 process 時是否產生資源衝突，當判斷出一個資源衝突，則演算法即可中止並回傳資訊。對使用者而言，能提供這樣的訊息是一個很好的服務。方法是首先判斷此 process 是否有參考的資源集合，若爲空集合可直接回傳 TRUE 表示不會產生

任何潛在資源衝突。接著可以忽略此新增 process 對所有 ancestor 或 descendant processes 的影響，如前節所述以回溯(back-tracking)方式逐步找出 AND-SPLIT 的 ancestor processes，然後逐一比較這個 AND-SPLIT 的其他分支上到與此新增 process 所處路徑的 AND-JOIN 會合前的各點，檢驗是否與此新增的 process 有資源交集，若有則表示產生潛在資源衝突。

```
Boolean OBD(workflow specification ws, process n)
1 {
2  // R(n) returns resource set that accessed by n
3  if (n.TYPE≠ACTIVITY) or (R(n)=ϕ) then return TRUE;
4
5  process set AND-JOIN_SET=ϕ;  // descendant AND-JOINs of n
6  flow queue P=ϕ;  // flows of parallel paths of n
7  flow queue Q=ϕ;  // in-flows of n for back tracking
8  insert all in-flows of n in ws into Q;
9
10 //PART-1: Finding n's descendants which are AND-JOIN
11 insert all out-flows of n into P;
12 while P≠ϕ
13 {
14    remove the first flow f from P
15    n' = sink of f;
16    if (n'.TYPE=AND-JOIN) then
17        add n' into AND-JOIN_SET;
18    insert all out-flows of n' into P;
19 }
20 set P = empty;
21
22 //PART-2: Finding all AND-Split parallel processes of n
23 while Q≠ϕ
24 {
25    remove the first flow f from Q;
26    n' = source of f;
27    if (n'.TYPE=AND-SPLIT) then
```

```
28        for each out-flow f'≠ f of n'
29        {
30            n" = sink process of f';
31            if Reachable(n",n)=FALSE then
32                insert f' into P;
33        }
34    insert all in-flow of n' into Q;
35 }
36
37 //PART-3: Detecting resource dependency
38 while P≠ϕ
39 {
40    remove the first flow f from P;
41    n" = sink process of f;
42    if (n".TYPE=ACTIVITY) and (R(n") ∩R(n) ≠ϕ) then
43        return FALSE;
44    if (n"∉AND-JOIN_SET) then insert all out-flows of n" into P;
45 }
46 return TRUE;
47}
```

Algorithm 5 Operation based detection

上述演算法當中 PART-1 的部分目的在搜尋並記錄插入點的 descendant processes 當中屬於 AND-JOIN 類型者，這些 processes 將作爲平行路徑上判斷資源衝突的一個界限。由於在稍後的演算法討論中也會運用到，在此將之獨立爲一個 COLLECT_AND-JOIN(ws, n)的副函式，回傳爲一個 process set。PART-2 則以相似的方法蒐集插入 process 的 ancestor processes 當中屬於 AND-SPLIT 者，然後將那些 AND-SPLIT 的分支當中，與插入 process 所處路徑平行的 flows，這些判斷可以副函式 COLLECT_Parallel-Path(ws, n)表示，並回傳一個 flow set。

接下來的演算法更進一步將一個新增插入的 process 所造成的所有影響，

一一的捕捉、列舉出來，這些訊息包括與其衝突的 processes 及發生衝突的資源項目。相較於 **Algorithm 5**，以下的演算法所提供資源衝突的詳盡訊息，對使用者而言是更好的服務。

```
void CPV(workflow specification ws, process n)
1 {
2  // R(n) returns resource set that accessed by n
3  if (n.TYPE≠ACTIVITY) or (R(n)=ϕ) then exit(0);
4  process set AND-JOIN_SET=ϕ;  // descendant AND-JOINs of n
5  flow queue P=ϕ;  // flows of parallel paths of n
6
7  AND-JOIN_SET =  COLLECT_AND-JOIN(ws,n);
8  P = COLLECT_Parallel-Path(ws,n);
9
10 while P≠ϕ
11 {
12    remove the first flow f from P;
13    n" = sink process of f;
14    if (n".TYPE=ACTIVITY) and (R(n") ∩R(n) ≠ϕ) then
15    {
16        for each resource r both in R(n") and R(n)
17            // List detailed information to user
18            printf("potential resource conflict between", n",
                      " and ", n, " with resource ", r);
19    }
20    if (n"∉AND-JOIN_SET) then insert all out-flows of n" into P;
21 }
22}
```

Algorithm 6 Conflict process verification

當增加某項資源於一個 activity process 上時，可利用類似 **Algorithm 5** 及 **Algorithm 6** 進行檢查，差別只是在比較資源相依性時僅考慮特定的一項資源 r，即上述兩個演算法當中對於 (R(n") ∩R(n) ≠ϕ) 的判斷改爲 (r∈R(n"))。

當刪除一個 activity process 或刪除一個 activity process 當中的一項資源時，利用前述的兩個互動式資源衝突檢驗演算法，也可在真正執行刪除動作前，偵測已存在的資源衝突，做爲告知使用者哪些會消除的資源衝突資訊，然後執行刪除動作。思考上述兩項互動式檢驗分析的方法，若能將以各 process 使用資源的情況儲存於 SPLIT process 上，則利用這些背景知識將有助於快速分析編輯操作，避免在分支流上逐點比對的工作。下一節將介紹如何利用這些記錄當作背景資訊加速資源衝突檢驗分析。

接下來討論關於 **Algorithm 6** 的時間複雜度，所有的 process 至多拜訪一次，因此在 worst case 的情況下，即拜訪工作流程規格中所有的 process，時間複雜度爲 O(n)。

3.4 利用分支路徑的資源串列達成遞增式分析

經由前述互動式的分析方法，雖然縮小資源檢驗所需考量的範圍，但尚未能有效的提升檢驗效率。本節介紹如何搭配分支路徑的資源串列記錄的方式達成遞增式資源衝突的分析。

首先針對工作流程規格中的 Control Block、分支路徑(SPLIT_PATH)，與分支路徑上的資源串列(resource list)做以下的定義：

Definition 6 (Control Block)

B = < n_s **,** n_e **> is a Control Block**

B.start = n_s **,** $n_s \in$ **N, B.end =** n_e **,** $n_e \in$ **N**

n_s **.TYPE = AND_SPLIT if and only if** n_e **.TYPE = AND-JOIN**

or n_s **.TYPE = XOR_SPLIT if and only if** n_e **.TYPE = XOR-JOIN**

$\forall$ **path** p_s **= <** n_s **,** n_1 **,** n_2 **,…,** n_k **,** n_e **>,** n_s **,** n_e **,** $n_i \in$ **N,** $i = 1,2,...,k$

for each p_s**, we call such path a SPLIT_PATH**

$\forall$ n_i **,** n_i **.TYPE=AND_SPLIT there exists an corresponding process** n_j **such that** $i < j \le k$ n_j **.TYPE=AND_JOIN**

$\forall$ n_i **,** n_i **.TYPE=XOR_SPLIT there exists an corresponding process** n_j **such that** $i < j \le k$ n_j **.TYPE=XOR_JOIN**

Definition 7 (Resource List on the SPLIT_PATH)

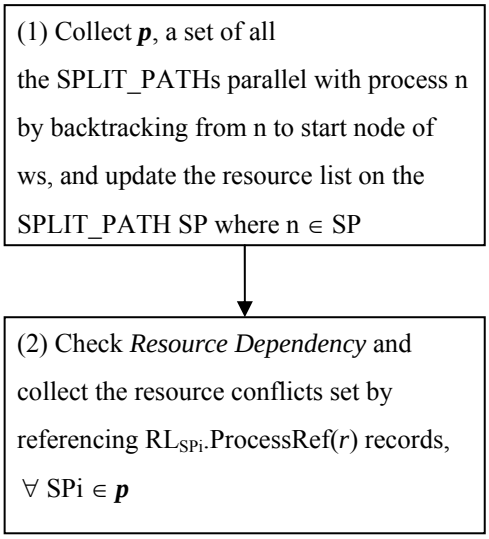
RL$_{SPi}$ is a resource list on the SPLIT_PATH SP$_i$, where

$\forall$ $n_j \in$ **SP$_i$,** **RL$_{SPi}$ =** $\cup$ R_j

$\forall$ r **,** r **is a resource, RL$_{spi}$.ProcessRef(**r**) ={**n_k **|** $n_k \in$ **SP$_i$,** $r \in R_k$ **}**

由以上的定義，接下來探討當 process *n* 上的一項資源做增刪操作，欲檢驗是否有資源一致性的改變，可利用類似 **Algorithm 6** CPV 中對 n 做回溯式的搜尋，當找到一個 AND-SPLIT，首先更新此 AND-SPLIT 對於 n 所處分支路徑(SPLIT_PATH)上的資源串列記錄，然後其他的分支路徑存於一個集合當中，最後比較此集合中的各個分支路徑(SPLIT_PATH)對應的資源串列記錄，如此避免掉在工作流程的控制流結構當中進行 traverse。假設編輯操作爲 process n 中某項資源 r 的新增，則以下圖的流程來說明稍後將介紹的遞增式演算法。其中第一個區塊當中(1)的部分爲搜集與 n 處在 AND-SPLIT 平行路徑上的分支路徑(SPLIT_PATH)，此條件可視爲造成資源衝突的第二條件，接著更新資源串列記錄，最後(2)利用(1)所回傳出的分支路徑，判斷是否資源相依，亦即造成資源衝

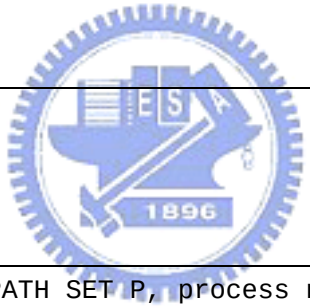
突的第一個條件，以回傳出正確的資源衝突集合。



將上述流程圖中 (1) 的部分獨立爲副函式 **CSP** (Collect SPLIT_PATHs)，並回傳一個 SPLIT_PATH 的集合，而 (2) 的部分則獨立爲副函式 **CRD** (Check Resource Dependency)，回傳的是一個資源衝突集合 (CONFLICT SET)，其中每個元素描述哪項資源對應到哪兩個 process 的一項資源衝突。其詳細內容撰寫如下：

```
SPLIT_PATH SET CSP(workflow specification ws, process n, resource r)
1 {
2  SPLIT_PATH SET P=ϕ;  // to store parallel AND-SPLIT paths of n
3  flow queue Q=ϕ;  // in-flows of n for back tracking
4
5  insert all in-flows of n in ws into Q;
6  // Finding all parallel SPLIT_PATHs with n
7  while Q≠ϕ
8  {
9     remove the first flow f from Q;
10    n' = source process of f;
```

```
11     if (n'.TYPE=AND-SPLIT) then
12     {
13         ∃ SPk such that n', n ∈ SPk;
14         RLSPk.ProcessRef(r)=RLSPk.ProcessRef(r) + n;//update records
15         for each out-flow f'≠ f of n'
16         {
17             n" = sink process of f';
18             if Reachable(n",n)=FALSE then
19             {
20                 ∃ SPm such that n', n" ∈ SPm;
21                 insert SPm into P;
22             }
23         }
24     }
25     insert all in-flow of n' into Q;
26 }
27   return P;
28}
```



```
CONLICT SET CRD(SPLIT_PATH SET P, process n, resource r)
1{
2   CONFLICT SET RC=ϕ; // record resource conflicts set
3   // Checking resource dependency
4   for each SPLIT_PATH SPi ∈ P
5      for each nj ∈ RLSPi.ProcessRef(r)
6          insert (r, n, nj) into RC;
7
8  return RC;
9}
```

綜合以上兩個副函式，則可將遞增式的演算法 IAC (Incremental Analysis for Conflicts) 描述如下：

```
void IAC(workflow specification ws, process n, resource r)
1 {
2  SPLIT_PATH SET P=ϕ;  // to store parallel AND-SPLIT paths of n
3  CONFLICT SET RC=ϕ;    // resource conflicts set
4
5  // Finding all parallel SPLIT_PATHs with n
6  P = CSP(ws, n, r);
7  // Detecting resource dependency
8  RC = CRD(P, n, r);
9
10 // showing information to users
11 for each resource conflict (r, n, nj) in RC
12    printf("There is a resource conflict ", (r, n, nj));
13}
```

Algorithm 7 Incremental analysis for conflicts

以上是描述新增加一項資源時的檢驗演算法，在實作資源串列時採用雙向鏈結配合雜湊表，則查詢資料串列時僅需 O(1)時間，效能僅與回溯長度有關，以 **Figure 7** 回溯長度最長的 worst case 而言，整體的時間複雜度爲 O(n)，雖然看似與互動式分析的時間複雜度一樣，但實際上對遞增式分析而言，此 O(n)表示回溯的 processes 數，並不等於需要計算的 processes 數，需要計算的部分僅是回溯路徑上的 AND-SPLIT 個數。詳細的時間複雜度討論將留待第五章當中。

Chapter 4. 考慮時間因素的資源分配限制分析

第三章的分析工作，並未加入時間因素的考量。實際上，若是工作流程規格在定義時，無法確切得知所有 activity process 的工作時間區段的話，就無法加入時間因素的分析，只能採用第三章的分析方法去判斷一個 activity process 的異動。本章假設各 activity process 的最小完成工作時限、最大完成工作時限 (min. and max. time durations) 能夠在設計階段時清楚定義，且 control processes 與 flows 無需消耗額外的時間的前提下，對 activity process 編輯分析加入時間因素的考量。

4.1 考慮時間因素的 activity process 編輯與影響

對於 activity processes 可編輯的操作而言，就會產生以下五種情況： (1) 新增插入 activity process， (2) 刪除 activity process， (3) 增加 activity process 某項參考的資源， (4) 刪除了 activity process 某項資源， (5) 修改 activity process 工作時限 (durations)。

考慮時間因素的情況下，使得(1)(2)的情況變得較複雜，因此將(1)(2)的討論置於修改 activity process 的工作時限內容之後討論。至於增加或是刪除已存在 activity process 的某項資源，其所帶來的影響分析，與不考慮時間因素的情況下，檢驗法相似，僅是多加入 EST 與 LET 的計算，根據 2.4.3 節中資源衝突定義的第三個條件，EAI overlapping，求出更精確的潛在資源衝突。

就更動工作時限而言，不僅會引發資源衝突的增加或消除，也可能會影響到其 descendent processes。假設一個被異動的 activity process a 原先的最小工作時限是 d, 最大工作時限 D, 經過編輯異動後最小工作時限改爲 d', 最大作時

限改爲 D’。在正常的改變下，意即 d’≤D’，則可分爲以下四種基本情況探討：

(1) D’ > D

a 的工作時限最大值增加，根據 2.4.3 節中 LET 的定義，意味著 LET(a) 延後，且所有 a 的 descendant processes 皆因此可能增加 LET 值。LET 的延後造成 EAI 變寬，須對這些受影響的 processes 做檢驗，以判斷新產生的資源衝突。這種影響會直到 End node，或者直到某一個 descendent process 的 LET 與 LET’不再有變化爲止。LET 不再有變化的情況乃是該 process 處在一個 AND-JOIN 或 XOR-JOIN 後，而原先產生 LET 變化的路徑在 JOIN 後不是最大值。

(2) D’ < D

a 的工作時限最大值減少，意味著 LET(a) 變小，根據定義，意味著所有 a 的 descendant processes 因此可能降低 LET 值，EAI 因此變窄，分析有哪些已存在的潛在資源衝突的情況因此消除。這種影響會直到 End node，或者直到某一個 descendent process 的 LET 與 LET’不再有變化爲止。

(3) d’ > d

a 的工作時限最小值增加，根據定義，意味著所有 a 的 descendant processes 因此可能增加 EST 值，EAI 因此變窄，需分析有哪些已存在的潛在資源衝突會因此消除。這種影響會直到 End node，或者直到某一個 descendent process 的 EST 與 EST’不再有變化爲止。

(4) d’ < d

a 的工作時限最小值減少，根據定義，意味著所有 a 的 descendant processes 因此可能降低了 EST 值，EAI 因此變寬，必須檢驗會有哪些新的潛在資源衝突情況發生。這種影響會直到 End node，或者直到某一個 descendent process 的 EST 與 EST’不再有變化爲止。

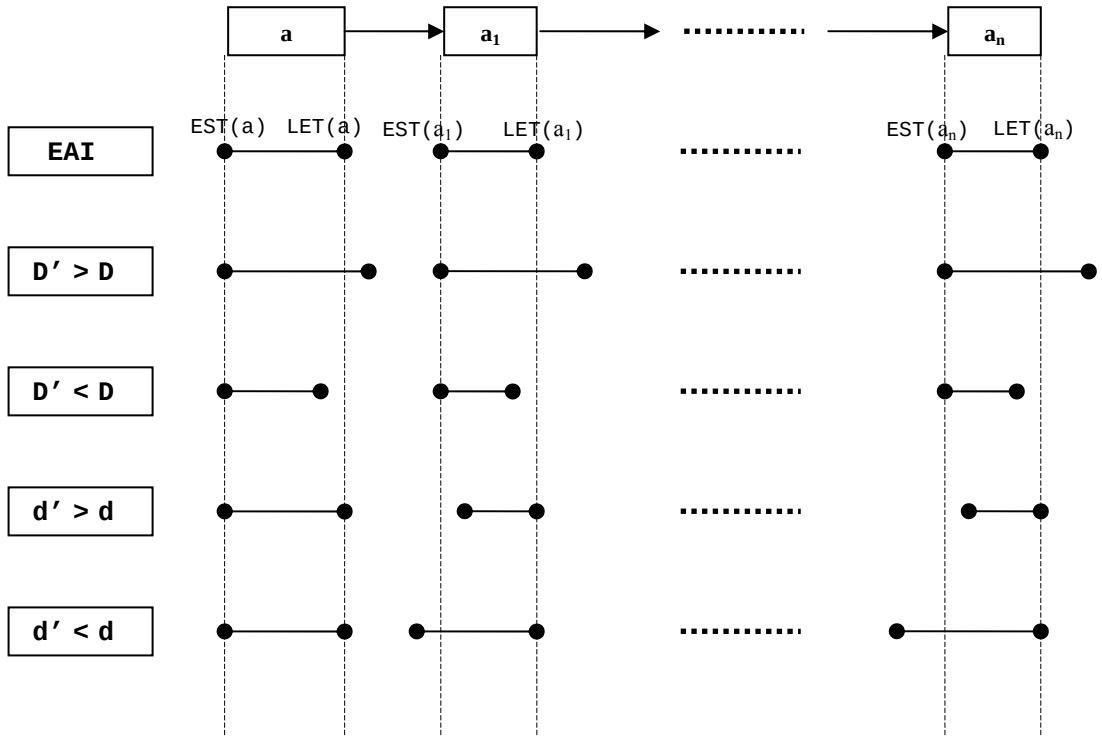


Figure 8 Changing durations of a process

因此可以根據上述規則，判斷 d → d' 與 D → D' 的變化，是否該做資源衝突的檢驗。**Figure 8** 用圖例來說明，工作時限異動對 activity process 以及其後的 descendant processes 所產生的連續影響。

當一個 activity process a 被新增、置入一個流程定義的結構當中，假設其 parent process 爲 a_0，child process 爲 a_1 。由於是新增的 process，其工作時限的設定 $(d(a) > 0, D(a) > 0)$ ，故勢必會造成 a_1 的工作時間區段受到影響，我們以 $EST'(a_1)$ 與 $LET'(a_1)$ 表示異動後 a_1 的最早開始工作時間與最晚工作結束時間，根據定義：

原先 $EST(a_1) = EST(a_0) + d(a_0)$

原先 $LET(a_1) = LET(a_0) + D(a_1)$;

插入 process a 在 a_0 與 a_1 間後：

$EST'(a_1) = EST(a) + d(a) = EST(a_0) + d(a_0) + d(a) = EST(a_1) + d(a) > EST(a_1)$

$LET'(a_1) = LET(a) + D(a_1) = LET(a_0) + D(a) + D(a_1) = LET(a_1) + D(a) > LET(a_1)$

除了 a 本身有可能造成新的潛在資源衝突外，由上述式子可以發現 a_1 的 EST 延遲了 d(a)，而其 LET 延遲了 D(a)。如此連鎖效應，對所有 descendant processes 的工作時間區段都產生兩種影響，第一種影響是這些 descendant processes 的 EST 皆延遲了 d(a)，可能因此消除了某些已存在的資源衝突，而第二種影響延遲了最晚工作結束時間，也就是說多出來的 D'(a) – D(a) 的這段時間就可能與某個 activity process 產生了潛在資源衝突。當新增一個 activity process a 時，除了 a 本身的檢驗外，由於影響了接續其後的 descendant processes，檢驗工作必須逐個執行。

若是從工作流程規格的結構當中刪去一個 activity process a，則可視爲此 process 改變成爲一個空的 (null) process，也就是說，$d \rightarrow d'=0$ 與 $D \rightarrow D'=0$，假設其 parent process 爲 a_0，觀察其 child process a_1 的 EST' 與 LET' 的變化：

原先 $EST(a_1) = EST(a) + d(a) = EST(a_0) + d(a_0) + d(a)$

原先 $LET(a_1) = LET(a) + D(a_1) = LET(a_0) + D(a) + D(a_1)$

刪除 process a 於 a_0 與 a_1 間後：

$EST'(a_1) = EST(a_0) + d(a_0) < EST(a_1)$

$LET'(a_1) = LET(a_0) + D(a_1) < LET(a_1)$

EST' (a_1) 的提早，造成 descendant processes 的工作時間區段可能與其他 processes 發生潛在資源衝突，LET'(a_1) 的提早也可能消除一些既有的資源衝突。而 a 後的所有 descendant processes 都會受到這種連鎖效應的影響，因此刪除 activity process，也要對其後接續的所有 processes 進行資源衝突增減的分析。

4.2 考慮時間因素的互動式檢驗演算法

考慮時間因素的情況下，3.3 節所描述的兩個主要的互動式演算法，皆需加入 EST 與 LET 的比對，以更進一步精確確認資源衝突，底下兩個演算法描述這些改變。由於插入新增 activity process 所帶來的影響不僅僅是該 process，且會影響到其所有後續 processes，必需以遞迴方式加入對後續 processes 的檢驗。

```
Boolean T_OBD(workflow specification ws, process n)
1 {
2  flow queue P=ϕ; // flows of parallel paths of n
3  if (n.TYPE=ACTIVITY) and (R(n)≠ϕ) then
4  {
5     P = COLLECT_Parallel-Path(ws,n);
6     while P≠ϕ
7     {
8         remove the first flow f from P;
9         n" = sink process of f;
10        if (LET(n) <= EST(n")) then
11            skip to next iteration;
12        else if (LET(n") <= EST(n)) then
13            insert all out-flow of n" into P;
14        else if (n".TYPE=ACTIVITY) and (R(n")∩R(n)≠ϕ) then
15            return FALSE;
16        else
17            insert all out-flow of n" into P;
18    }
19  }
20  for each child process n' of n
21    if (T_OBD(ws,n')=FALSE) then return FALSE;
22 return TRUE;
23}
```

Algorithm 8 Operation based detection with temporal consideration

```
void T_CPV(workflow specification ws, process n)
1 {
2  flow queue P=ϕ; // flows of parallel paths of n
3  if (n.TYPE=ACTIVITY) and (R(n)≠ϕ) then
4  {
5     P = COLLECT_Parallel-Path(ws,n);
6     while P≠ϕ
7     {
8         remove the first flow f from P;
9         n″ = sink process of f;
10        if (LET(n) <= EST(n″)) then
11            skip to next iteration;
12        else if (LET(n″) <= EST(n)) then
13            insert all out-flow of n″ into P;
14        if (n″.TYPE=ACTIVITY) and (R(n″)∩R(n)≠ϕ) then
15        {
16            for each resource r both in R(n″) and R(n)
17                // List detailed information to user
18                printf ("Potential resource conflict between", n″,
                        "and", n, "with resource", r);
19        }
20        else
21            insert all out-flow of n″ into P;
22    }
23 }
24 for each child process n′ of n
25    T_CPV(ws,n′)
26}
```

Algorithm 9 Conflict process verification with temporal consideration

當增加／刪除一個 activity process n 上的某一項資源 r 時，可利用類似 T_OBD 及 T_CPV 分別進行有無資源衝突產生與產生哪些資源衝突的檢查，差別僅是比較資源相依性時僅考慮此該項資源 r 即可，即上述兩個演算法當中對於 (R(n″) ∩R(n) ≠ϕ) 的判斷改爲 (r∈R(n″))。此時的檢驗判斷便無需考慮 n 之後的 descendant processes，因爲改變一個 process 上的資源參考，並不會引起

其後續 process 關於資源衝突方面的變化。

若是修改內容包含工作時限 (duration) 的最大值、最小值的修改，這些改動影響到一個 activity process 的最大可能工作區段，根據 **Figure 8** 中的分析，process 的可能工作區段因爲這些改動而超出原有區段的話，就可能與其他 processes 工作區段發生重疊。

以下的演算法 T_DM (Temporal analysis for duration modification) 檢驗當一個 activity process n 的工作時限改變時，此演算法可檢驗造成 n 及其 descendent processes 相關的資源衝突。

```
void T_DM(workflow specification ws, process n, d', D')
1 {
2  if D'(n)>D(n) then  // checking n & n's descendants
3     T_CPV(ws,n);
4  else if d'(n)<d(n) then  // checking n's descendants
5  {
6     for each child process n' of n
7         T_CPV(ws,n');
8  }
9 }
```

Algorithm 10 Temporal analysis for duration modification

4.3 EAI 時間表

當工作流程規格考量時間因素的情況下，除了能利用 3.4 節當中介紹的分支路徑資源串列記錄來輔助外，此節介紹的 EAI 時間記錄表，可用來記錄所有 processes 的 EST 與 LET 資料。這些記錄的維護是當使用者對工作流程規格有新增、刪除 process，或改變 process 的工作時限 (durations)，則更新受到影響的各個欄位內容。利用這個記錄表就不必等到檢驗時才在每個檢查迴圈當中從 Start node 重新計算所有 processes 的 EST 與 LET 值，由此加速檢驗時的效率。在稍後第五章的劇本範例中將有使用 EAI Table 方法的解說。

4.4 考慮時間因素的遞增式分析

承續 4.2 節最後 **Algorithm 10** 關於 durations 改變的討論，可以發現互動式演算法僅能列出現有的資源衝突，但未能精確的指出一次編輯所新增或消除掉哪些資源衝突。在此將各種 activity process n_x 的編輯狀況列入一個表中，並加以說明，以方便稍後描述遞增式演算法時採用。表格當中 RC 代表資源衝突的數量：

Operation on n_x	**n_x**	**n_{xi}, descendant of n_x**
$d'(n_x)>d(n_x)$ **$D'(n_x)>D(n_x)$**	$LET'(n_x) \uparrow \rightarrow RC \uparrow$	$EST'(n_{xi}) \uparrow \rightarrow RC \downarrow$ $LET'(n_{xi}) \uparrow \rightarrow RC \uparrow$
$d'(n_x)>d(n_x)$ **$D'(n_x)<D(n_x)$**	$LET'(n_x) \downarrow \rightarrow RC \downarrow$	$EST'(n_{xi}) \uparrow \rightarrow RC \downarrow$ $LET'(n_{xi}) \downarrow \rightarrow RC \downarrow$
$d'(n_x)<d(n_x)$ **$D'(n_x)>D(n_x)$**	$LET'(n_x) \uparrow \rightarrow RC \uparrow$	$EST'(n_{xi}) \downarrow \rightarrow RC \uparrow$ $LET'(n_{xi}) \uparrow \rightarrow RC \uparrow$

$d'(n_x)<d(n_x)$ **$D'(n_x)<D(n_x)$**	$LET'(n_x)\downarrow \rightarrow RC\downarrow$	$EST'(n_{xi})\downarrow \rightarrow RC\uparrow$ $LET'(n_{xi})\downarrow \rightarrow RC\downarrow$

Table 1 Temporal operations on process n_x

觀察此表當中 n_x 的 LET'下降時，伴隨著資源衝突數量的下降，這些減少的資源衝突，可先用分支路徑的資源串列記錄，快速查出針對 process n_x 已存在的資源衝突集合，針對當中各元素(r, n_x, n_i)進行下列判斷式：

$$[EST(n_x), LET(n_x)] \cap [EST(n_i), LET(n_i)] \subseteq [LET'(n_x), LET(n_x)]$$

檢查若上述式子成立時，則 (r, n_x, n_i) 爲一組會被消除的資源衝突。至於 n_{xi} 的 LET 改變時可比照相同方法進行分析比較，直到 End node 或某個 descendant process n_{xj} 的 $LET'(n_{xj}) = LET(n_{xj})$，此時意味著 n_x 所影響到的 path 無法提供 $LET(n_{xj})$ 的最大值，而檢驗分析也可到此爲止，因爲不再影響到 n_{xj} 之後的 descendant processes。上述判斷式當中 EST/LET 值都可透過 EAI 時間表來查詢，而修改的部分僅需重新計算 $LET'(n_x)$，因此可對這個判斷式進行快速的檢查。

當 n_{xi} 的 EST'延遲，則可能帶來部分資源衝突消除，下列判斷式可檢查 (r, n_{xi}, n_i) 的已存在衝突是否被消除：

$$[EST(n_{xi}), LET(n_{xi})] \cap [EST(n_i), LET(n_i)] \subseteq [EST(n_{xi}), EST'(n_{xi})]$$

同樣的，當依據上述判斷式依序判斷 n_{xi} 的 descendant processes，直到 End node 或某個 descendant process n_{xj} 的 $EST(n_{xj}) = EST'(n_{xj})$，此時意味著 n_x 所影響到的 path 無法提供 $EST(n_{xj})$ 的最小值，而檢驗分析也可到此爲止，因爲不再影響到 n_{xj} 之後的 descendant processes。

至於 LET'(n_x)與 LET'(n_x)的延遲，造成新增的資源衝突，僅需根據以下檢驗式來進行檢查。其中 n_j 爲相對於 n_x，符合資源衝突定義中第一與第二條件者：

$$\mathbf{[LET(n_x), LET'(n_x)] \cap [EST(n_j), LET(n_j)] \neq \phi}$$

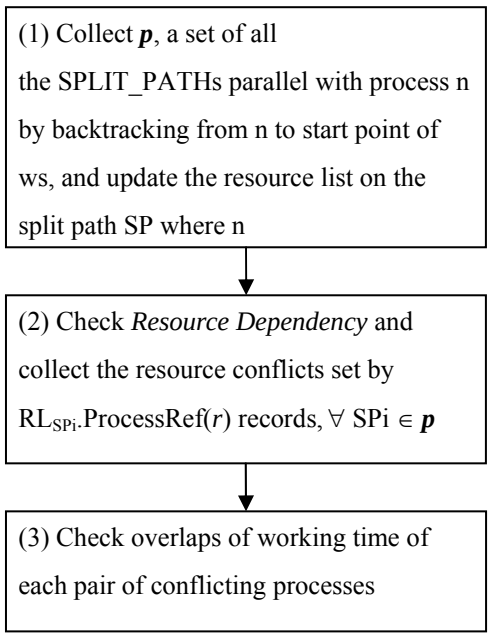
依據上述判斷式依序判斷 n_x 的 descendant processes n_{xi}，將 n_{xi} 代入 n_x 直到 End node 或某個 descendant process n_{xj} 的 LET(n_{xj}) = LET'(n_{xj})，此時意味著 n_x 所影響到的 path 無法提供 LET(n_{xj}) 的最大值，檢驗分析可到此爲止，因爲不會影響到 n_{xj} 之後的 descendant processes。

最後分析 EST'(n_{xi})的提前，造成新增的資源衝突，則根據以下檢驗式來進行檢查。其中 n_j 爲相對於 n_x，符合資源衝突定義中第一與第二條件者：

$$\mathbf{[EST'(n_{xi}), EST(n_{xi})] \cap [EST(n_j), LET(n_j)] \neq \phi}$$

根據上述判斷式依序判斷 n_{xi} 的 descendant processes，直到某個 descendant process n_{xj} 的 EST(n_{xj}) = EST'(n_{xj})，此時意味著 n_x 所影響到的 path 無法提供 EST(n_{xj}) 的最小值，而檢驗分析也可到此爲止，因爲不再影響到 n_{xj} 之後的 descendant processes。

上述四個檢驗式，目的在於利用已知的背景知識，快速比對出編輯修操作所造成哪些資源衝突新的增或消去，而非使用 T_CPV 互動式的方法以資源衝突的第三個定義作判斷。如同 3.4 節當中描述，以下先以流程圖來描述考慮時間因素下的遞增式分析：



(1)(2)的演算法與 3.4 節中的討論相同。在此僅描述(3)的演算法 CTO (Checking Time Overlapping) 如下，從(2)傳入的資源衝突集合，做一個篩選比對，符合時間重疊的判斷式時予以留下，否則就從集合中刪除該項資源衝突。

```
void CTO(conflict set RC, EXP TIME_EXP)
1 {
2  // Checking working time overlapping with expression EXP
3  for each resource conflict (r, n, ni) in RC
4     if not (TIME_EXP) then
5         remove (r, n, ni) from RC;
6 }
```

此處的判斷式暫以 EXP 表示，而不以 2.4 節的 EAI 重疊的定義來做檢驗，爲的是保留這個判斷式的彈性以符合本節所討論的 EST 與 LET 的提前、延後所造成資源衝突的影響。EXP 的類型包括以下幾種，其中 n 表示受到編輯操作的 process，而 n_i 爲來自(2)的集合中與 n 成對的衝突 process：

(a) **[EST(n), LET(n)]∩[EST(n_i), LET(n_i)] ≠ ϕ**

(b) **[EST(n), LET(n)]∩[EST(n_i), LET(n_i)]⊆[LET'(n), LET(n)]**

(c) **[EST(n), LET(n)]∩[EST(n_i), LET(n_i)]⊆[EST(n), EST'(n)]**

(d) **[LET(n), LET'(n)]∩[EST(n_i), LET(n_i)] ≠ ϕ**

(e) **[EST'(n), EST(n)]∩[EST(n_i), LET(n_i)] ≠ ϕ**

至於 EST 與 LET 值的計算透過 4.3 節中所描述的 EAI 時間記錄表，可避免每次存取都需從 Start node 開始累加計算。

考慮時間因素的遞增式演算法便可由 IAC 改爲如下的 T_IAC。當要用於分析新增／刪除 process n 上的一項資源 r 時，只需將 EXP E = "[EST(n), LET(n)]∩[EST(n_i), LET(n_i)] ≠ ϕ " 代入其中即可：

```
void T_IAC(workflow specification ws, process n, resource r, EXP E)
1 {
2  SPLIT_PATH SET P=φ; // to store parallel AND-SPLIT paths of n
3  CONFLICT SET RC=φ;    // resource conflicts set
4
5  // Finding all parallel SPLIT_PATHs with n
6  P = CSP(ws, n, r);
7  // Checking resource dependency
8  RC = CRD(P, n, r);
9  // Checking working time overlapping
10 CTO(RC, E);
11
12 // showing information to users
13 for each resource conflict (r, n, ni) in RC
14    printf("There is a resource conflict ", (r, n, ni));
15}
```

接下來根據 Table 1 中考慮各種改變 duration 操作，其遞增式演算法 T_CDM (Checking Duration Modification)爲：

```
void T_CDM(workflow specification ws, process n, EST', LET')
1 {
2  CONFLICT SET RC=ϕ;    // resource conflicts set
3
4  if LET'(n)<LET(n) then  // it may reduce resource conflicts
5  {
6   for each r ∈ R(n)
7      RC=RC+T_IAC(ws, n, "[EST(n),LET(n)]∩[EST(ni),LET(ni)]⊆[LET'(n),LET(n)]");
8   for each resource conflict (ri, n, nj) in RC
9      printf("resource conflict ", (ri, n, nj), " is removed.");
10 }
11 else if LET'(n)>LET(n) then // it may increase resource conflicts
12 {
13    for each r ∈ R(n)
14        T_IAC(ws, n, r, "[LET(n),LET'(n)]∩[EST(ni),LET(ni)] ≠ϕ");
15 }
16
17 if EST'(n)<EST(n) then // it may increase resource conflicts
18 {
19    for each r ∈ R(n)
20        T_IAC(ws, n, r, "[EST'(n),EST(n)]∩[EST(ni),LET(ni)] ≠ϕ");
21 }
22 else if EST'(n)>EST(n) then // it may reduce resource conflicts
23 {
24    for each r ∈ R(n)
25      RC=RC+T_IAC(ws, n, "[EST(n),LET(n)]∩[EST(ni),LET(ni)]⊆[EST(n),EST'(n)]");
26    for each resource conflict (ri, n, nj) in RC
27    printf("resource conflict ", (ri, n, nj), " is removed.");
28 }
29 if (EST'(n)≠EST(n)) or (LET'(n)≠LET(n)) then
30    for each child process n' of n
31        T_CDM(ws,n');
32}
```

而當新增／刪除 activity process n 時該採取的分析方法，則類似前述兩個演算法的組合運用，針對 n 本身的各項資源參考進行如同新增／刪除一項資源的檢驗，而對其 descendent processes 則以 T_CDM 遞回的方式檢查那些 EST 與 LET 受改變的範圍。詳細內容描述如下：

```
CONFLICT SET T_IAC_ACTIVITY(ws, n)
1 {
2  CONFLICT SET RC=ϕ;    // resource conflicts set
3
4  for each rj ∈ R(n)
5     T_IAC(ws, n, rj,"[EST(n),LET(n)]∩[EST(ni),LET(ni)] ≠ϕ" )
6
7  for each child process n' of n
8     T_CDM(ws,n');
9 }
```

最後將考慮時間因素的情形下，將以上三種演算法做一個總結 T_OBIAC (Operation based incremental analysis for conflict with temporal consideration，根據操作類型來處理的遞增式資源衝突分析) 如下：

```
void T_OBIAC(workflow specification ws, process n, operation OP)
1 {
2  CONFLICT SET RC=ϕ;    // resource conflicts set
3
4  if (OP="ADD OR REMOVE A RESOURCE") then
5     T_IAC(ws, n, r);
6  else if (OP="CHANGING DURATIONS") then
7     T_CDM(ws, n, EST'(n), LET'(n));
8  else if (OP=" ADD OR REMOVE AN ACTIVITY") then
9     T_IAC_ACTIVITY(ws, n);
10}
```

Algorithm 11 Operation based incremental analysis with temporal consideration

Chapter 5. 討論與範例

5.1 時間複雜度討論

	Batch	Mod. Batch	Interactive	Incremental
worst case	$O(n^3)$	$O(n^2)$	O(n)	O(n)

Table 2 Time complexity comparison

由先前的討論中，各個演算法在 worst case 的時間複雜度列表如上。當中遞增式演算法雖然 tracking 的 processes 數與互動式演算法都是逼近 O(n)，但遞增式演算法僅在 CONTROL BLOCK 的分支點爲 AND-SPLIT 時才會進行計算，若工作流程規格中的 AND-SPLIT 數量少時，計算也會相對很少。所以 tracking 的 processes 個數並不等於需要計算的 processes 個數。

從需要計算的觀點來看，遞增式演算法的計算量，會與 AND-SPLIT 個數成正比。考慮 **Figure 9** 僅一個 AND-SPLIT 的情況，假設編輯 process n_1 時採用遞增式演算法，僅需 O(1)的計算時間。若採用互動式演算法，則需拜訪所有其他的 processes，並存取計算這些 processes 對應的資源參照集合，時間花費 O(n)。

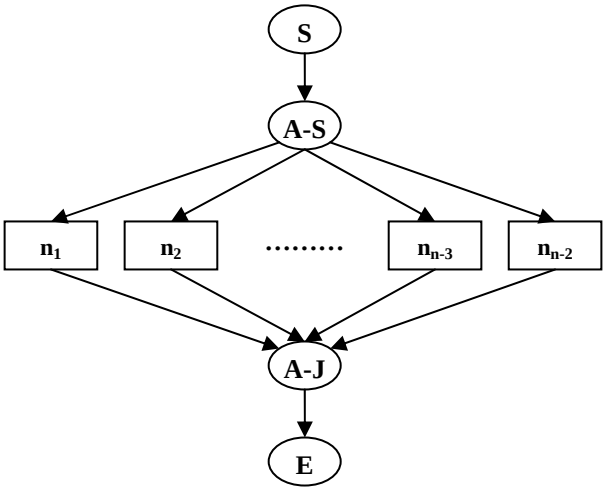


Figure 9 an example with only one AND-SPLIT

接著考慮當 AND-SPLIT 個數漸漸增多時的狀況，例如當接近 O(n)個 AND-SPLITs，則可用下圖的工作流程做爲例子來說明。假設 processes 總數爲 n 個，以類似兩個互相對稱的二元樹架構來看，可以發現自 Start node S 到 End node E 的路徑長約爲 2*log*(n/2)，若是在 process n_1 上進行編輯操作，遞增式演算法只計算 n_1 回溯到起始點的路徑上的每個 ancestor AND-SPLIT，因此時間複雜度大概會在 O(*log*(n/2))≈O(*log*n)。同樣地互動式演算法會拜訪並計算所有其他 n 個 processes 一次，時間複雜度爲 O(n)。

從計算量的觀點來說，可以保證遞增式演算法較互動式演算法爲優。

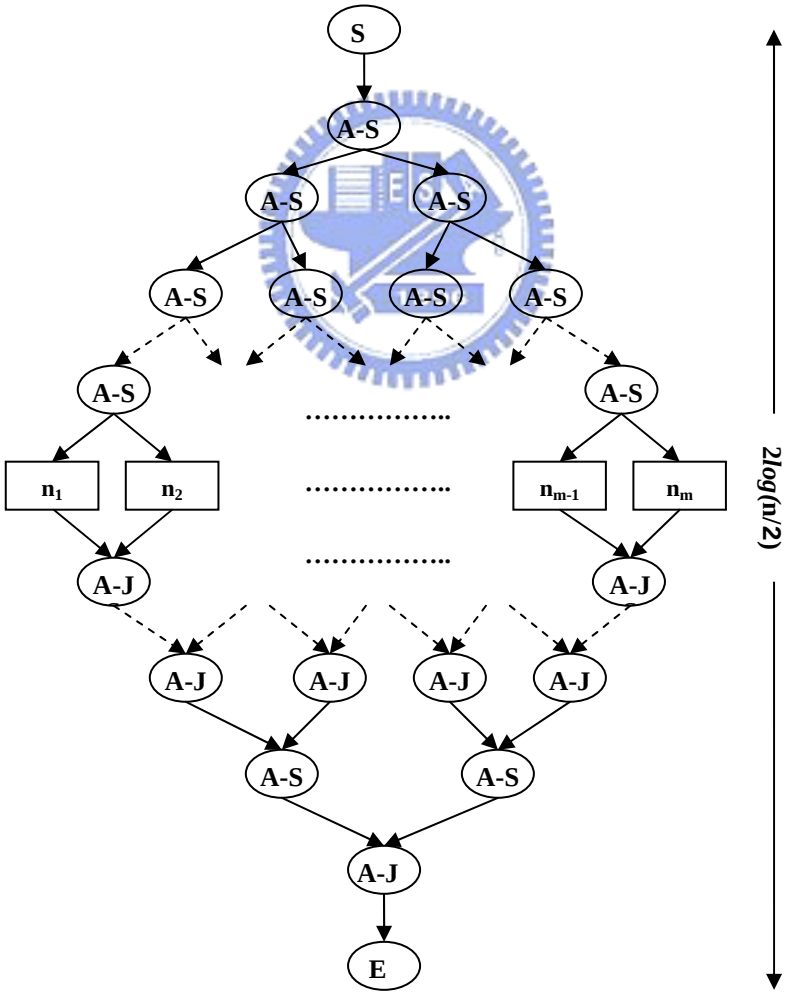


Figure 10 an example with lots of AND-SPLITs

5.2 劇本範例

本節以一個劇本範例來說明前幾節當中演算法的運作，此劇本範例包含時間因素的考量，下圖中表示已編輯好的情況，**Table 3** 與 **Table 4** 分別是分支路徑資源串列記錄表以及 EAI 時間記錄表，**Table 3** 中 $RLsp_1$ 與 $RLsp_2$ 爲儲存於 $A\text{-}S_1$ 上，記錄起於 $A\text{-}S_1$ 終於 $A\text{-}J_1$ 的兩條 SPLIT_PATH SP_1 及 SP_2 相對應的資源使用情況。$RLsp_3$ 與 $RLsp_4$ 則爲儲存於 $A\text{-}S_2$ 上，記錄起於 $A\text{-}S_1$ 終於 $A\text{-}J_1$ 的兩條 SPLIT_PATH SP_3 及 SP_4 相對應的資源使用情況。

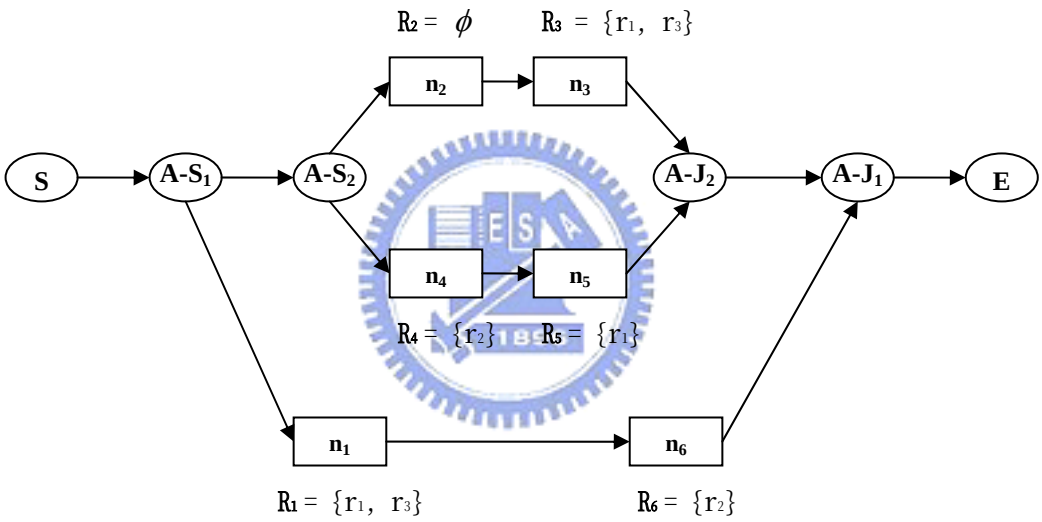


Figure 11 Scenario example

		r_1	r_2	r_3
$A\text{-}S_1$	$RLsp_1$	n_3, n_5	n_4	n_3
	$RLsp_2$	n_1	n_6	n_1
$A\text{-}S_2$	$RLsp_3$	n_3	-	n_3
	$RLsp_4$	n_5	n_4	-

Table 3 Resource lists on the split paths

	d	D	EST	LET
n_1	2	4	0	4
n_2	3	5	0	5
n_3	1	4	3	9
n_4	4	6	0	6
n_5	3	7	4	13
n_6	5	10	2	14

Table 4 EAI time table

接下來我們對這個工作流程規格進行 4.1 節當中所討論的幾項編輯操作，並分析說明所造成的影響：

(1) 新增插入一個 process n_7 於 n_1 與 n_6 間，且 $d(n_7)=5$, $D(n_7)=8$, $R(n_7)=r_2$：首先擴增 Table 4 加入 n_7 的計算，$EST(n_7)=2$，$LET(n_7)=12$；同時更動了其後續 process n_6 的 EST 與 LET 值，$EST'(n_6)=7>EST(n_6)=2$，$LET'(n_6)=22>EST(n_6)=14$，推斷可能部分的資源衝突被移除且產生一些新的資源衝突。由 **錯誤! 找不到參照來源。**判斷出 n_7 將帶來的新資源衝突，接著利用 T_CPV 演算法往前回溯追蹤，在 $A\text{-}S_1$ 的資源使用串列記錄中，首先更新自身所在的 $RLsp_2$ 的記錄，然後比對其他分支上的使用情況，由於 n_7 使用了 r_2 這項資源，從 $RLsp_1$ 上查出 r_2 亦爲所使用，緊接著從 Table 5 更新過後的 EAI 時間表上判斷，n_7 與 n_4 由於時間有重疊到，因此會出現資源衝突。接著由**錯誤! 找不到參照來源。**後半遞迴判斷 descendant processes n_6。n_6 回溯至 $A\text{-}S_1$ 得已存源衝突(r_2, n_4, n_6)，接著根據 Table 5 可計算出 $[EST(n_6), LET(n_6)] \cap [EST(n_4), LET(n_4)]=[2,6] \subseteq [2,7]=[EST(n_6), EST'(n_6)]$，因此($r_2$, n_4, n_6)這項資源衝突會因 n_7 插入而消除。最後考慮是否因 $LET'(n_6)$的延後產生新的資源衝突，從 Table 5 發現不存在與$[LET(n_6), LET'(n_6)]=[14, 22]$有交集的 process，因此不會產生新的資

源衝突。由於 n_6 後已無 activity process，檢查到此結束。下列各圖表示這次編輯後的結果。

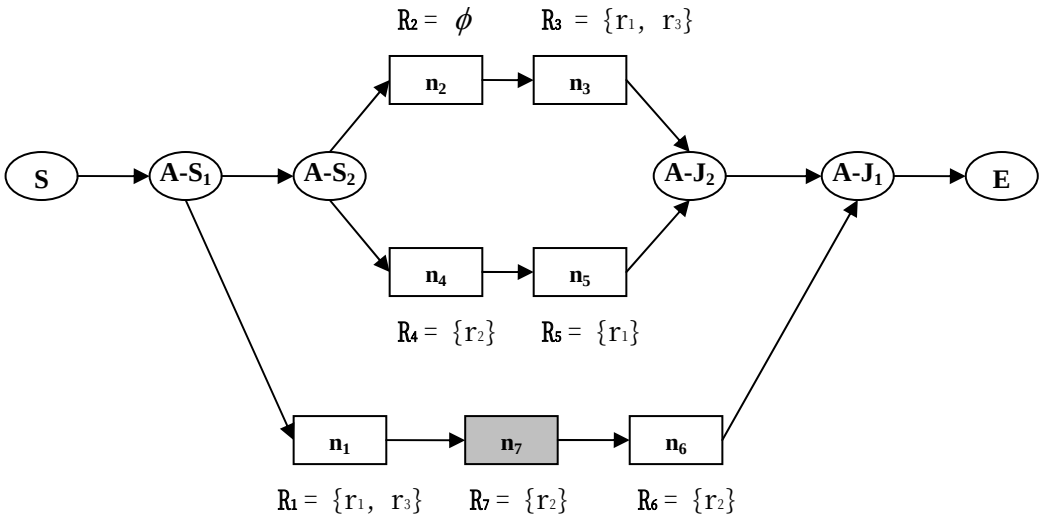


Figure 12 Inserting n_7 between n_1 and n_6

	d	D	EST	LET
n_1	2	4	0	4
n_2	3	5	0	5
n_3	1	4	3	9
n_4	4	6	0	6
n_5	3	7	4	13
n_6	5	10	**2→7**	**14→22**
n_7	**5**	**8**	**2**	**12**

Table 5 EAI time table

		r_1	r_2	r_3
A-S_1	**RLsp$_1$**	n_3, n_5	n_4	n_3
	RLsp$_2$	n_1	n_6, **n_7**	n_1
A-S_2	**RLsp$_3$**	n_3	-	n_3
	RLsp$_4$	n_5	n_4	-

Table 6 Resource lists on the split paths

(2) 當移除 process n_7 時，一樣的回溯到 A-S_1 中查詢哪些資源衝突會因此而消除。將訊息通知使用者後，接著進行 $RLsp_2$ 的維護，以及刪除 EAI 時間記錄表上相關的欄、列記錄。除此之外，在刪除 n_7 之後，也需對其後的後續 processes 的 EST 與 LET 值進行更新，並檢查其 descendent processes 是否因此增刪了資源衝突。此操作最後結果等同將 Table 5 還原爲 Table 3，Table 6 還原爲 Table 4。

(3) 將 process n_1 資源參考項目增加一項 r_2，則由 T_CPV 可以回溯至 A-S_1 查出所造成的資源衝突爲 (r_2, n_1, n_4)，將此資訊通知使用者，並將 Table 4 中 $RLsp_2$ 對應 r_2 的位置上資訊更新爲 n_1, n_6。

(4) 緊接著當消除(3)中 n_1 的參考項目 r_2，同樣地可以由 T_CPV 演算法回溯查詢比對 $RLsp_1$ 上 r_2 的使用情況而得知資源衝突 (r_2, n_1, n_4) 將消除。將 Table 4 中 $RLsp_2$ 對應 r_2 的位置上資訊更新爲 n_6。

(5) 改變 process n_4 的 durations：由 $d(n_4)=4 \rightarrow d'(n_4)=1$，$D(n_4)=6 \rightarrow D'(n_4)=2$。此變化造成了 $LET'(n_4)=2<LET(n_4)=6$，其後續 process n_5 也受到改變 $EST'(n_5)=1<EST(n_5)=4$，$LET'(n_5)=9<LET(n_5)=13$，將這些更新後的狀況記錄到 Table 8。此時根據回溯到 A-S_2 比對 $RLsp_3$ 上並無 n_4 所用資源 r_2 被其他 processes 參考，因此繼續回溯到 A-S_1 比較，得到 $RLsp_2$ 上的 n_4 與 n_6 間存在資源衝突(r_2, n_4, n_6)。接著根據 4.4 節的判斷式，$[EST(n_4), LET(n_4)] \cap [EST(n_6), LET(n_6)]=[2,6] \subseteq [2,6]=[LET'(n_4), LET(n_4)]$，可以判斷出資源衝突 ($r_2$, n_4, n_6) 會被消除。由於 EST 與 LET 的改變並未終止，因此需遞迴的考慮 n_4 的 descendent process n_5，同樣的回溯 A-S_2 與 A-S_1 上的資源串列記錄同時比對 EAI 時間記錄表，查得原本與 n_5 有資源衝突的爲 n_3，但$[EST(n_5), LET(n_5)] \cap [EST(n_3), LET(n_3)]=[4,9] \not\subset [9,13]=[LET'(n_5), LET(n_5)]$，所以已存在的資源衝突 ($r_1$, n_3, n_5)

無法消除。另外由於判斷到[EST'(n_5), EST(n_5)]∩[EST(n_1), LET(n_1)]=[1,4]≠ϕ，所以將產生新資源衝突 (r_1, n_1, n_5)。經過上述編輯異動，獲得以下 Table 8 與 Table 9 分別代表 EAI 時間記錄表與分支路徑資源串列記錄的最後狀態。

	D	**D**	**EST**	**LET**
n_1	2	4	0	4
n_2	3	5	0	5
n_3	1	4	3	9
n_4	**4→1**	**6→2**	0	**6→2**
n_5	3	7	**4→1**	**13→9**
n_6	5	10	2	14

Table 7 Updated abstract time table

		r_1	**r_2**	**r_3**
A-S_1	**RLsp$_1$**	n_3, n_5	n_4	n_3
	RLsp$_2$	n_1	n_6	n_1
A-S_2	**RLsp$_3$**	n_3	-	n_3
	RLsp$_4$	n_5	n_4	-

Table 8 Updated resource lists on the split paths

Chapter 6. 相關研究

一般設計工作流程，是由真實的商業流程加以抽象化後，再以工作流程規格語言來描述其定義，我們稱這樣的定義爲工作流程的規格 (workflow specification)。定義工作流程規格的過程是複雜而且容易導致錯誤的，特別是大型 (large-scale) 的系統。目前幾個工作流程分析的角度包括有：(1)工作流程檢驗 [21] (2)工作流程模擬 [22] (3)工作流程效能分析 [23]。其中工作流程檢驗目的是檢查工作流程規格有無缺陷，因爲工作流程規格的滿足正確性對工作流程執行而言極爲重要，且必須保持這樣的正確性才能達成商業的目標。[2]

6.1 工作流程規格的驗證

流程規格驗證目前主要以結構、時間、與資源三方面來分析，當中最基本也最重要的，當屬結構驗證，因爲工作流程規格必須保有其正確性，否則可能連執行都無法開始。目前已知的結構驗證包含 Petri-nets、有向圖、代數等等方法。

Petri-nets 將工作流程中的活動之間的傳遞與相依關係對映到 places 及 arcs，且利用既有的 Petri-nets 分析技術來分析工作流程規格的正確性與一致性。Sadiq 等人[3] 及 Onoda 等人[13] 利用有向圖分析工作流程規格，將隱含於規格當中可能造成死結 (deadlock) 或缺乏同步性 (lack of synchronization) 的狀況利用有效的演算法偵測出來。以代數爲基礎來驗證工作流程規格的方法，是將工作流程中的 activity 表示爲符號，而 activities 間的流向表示爲運算子。藉由分析這些數學運算式來判斷是否符合工作流程規格的正確性。Singh [14] 提出了事件代數 (event algebra) 來定義工作流程的模型，以便分析其正確性。

除了結構驗證被廣泛討論外，時間因素的考量也有諸多文獻提及。 1957 年 J.E. Kelly 與 M.R. Walker 共同開發了所謂的「關鍵路徑方法」(CPM, critical path method) [15]，用於製造生產與裝配作業的時間預估與控管，隨著應用領域擴大，目前也運用到一般商業流程上。根據修改過的關鍵路徑方法，可以在工作流程設計階段估算像是特定時間限制內要完成的工作、或是一些時間上限下限的限制。Eder 等人[16] 在 1999 年以工作流程圖形來定義出最早結束時間與最晚結束時間，Marjanovic [4] 則於 2000 年假設了每項 activity 有最小工作時限 (minimum duration) 與最大工作時限 (maximum duration)，並給予每個 activity process 在執行階段時起始時間與結束時間，藉此達到設計階段更精確的時間限制驗證。Zhunge 2001 [17] 則加入了不同時區、流向時間等考量，除了爲工作流程時間限制的一致性作更深入的分析，也巧妙利用不同時區的時差，將工作任務調整爲時間限制內可以完成。此外 Adam [10] 也運用考慮時間因素的 Petri-nets (TCPN) 在設計間段辨別工作流程時間上的恰當性。

驗證工作流程的正確性，並不止於結構與時間上。工作流程中的 processes 在操作時，需要存取各式各樣的資源來協助其完成作業，就現實環境而言，資源限制成了驗證工作流程正確性的一項重要課題。Hongchen Li 等人於 2004 年 [2] 當中討論資源發生不一致性所可能導致的問題，且對資源一致性驗證作分析並提供適當的演算法，最後更進一步的加入了時間因素的考量。然而這種完整且全面的檢驗分析，對於流程設計者而言，並未能達到即時性的輔助。

6.2 遞增式設計與需求

遞增式分析原本是微觀經濟學 (microeconomics) 上的一種分析方法，主要針對特定變因的微小改變，所造成相關變因與整體系統的影響，這種分析方法也稱作邊際分析 (marginal analysis)。

軟體工程開發上所討論的遞增式開發則是一項類似上述的重要技術。以這種技術開發軟體可以避免「大爆炸」(“Big Bang”) 效應，也就是說，長時間沒有發生任何狀況，但突然間，系統被導向一個從未面對過的新狀況。遞增式發展方法的觀念，是以軟體成長的方式取代軟體建設。同時遞增式開發方法，能將觀察優先聚焦於軟體本質特性，也就是說額外的功能只有再需要的時候才被加進來。[18]

針對軟體的設計而言，採用遞增式分析來觀察異動部份的元件在異動後所造成的局部影響，然後採取相對應的措施。避免設計者因爲小異動而仍須大規模檢驗其影響。此類遞增性檢驗分析，常見的應用如有：文書軟體的錯誤拼字檢查、程式編寫環境對指令與變數即時檢查等等。

Chapter 7. 結論與未來研究方向

工作流程規格是輔助工作流程自動化的正規化方法，許多工作流程管理系統針對工作流程的設計環境與執行環境，提供便利於設計者、使用者與管理者的工具。本文提出在設計階段一系列的互動式資源衝突檢驗分析討論，與改進後的遞增式檢驗方法，對於設計工作流程的設計師即時編輯而言，有絕對正面的幫助，除此之外也能夠有效降低執行階段的可能會遭遇的負擔與困擾。

已知的文獻中對於工作流程規格的結構正確性與時間一致性的驗證討論甚爲豐富，但對於資源驗證方面卻停留在完整而缺乏效率的方法。我們以遞增式設計的想法，分析設計者編輯 process 所帶來的局部影響，經過適當與適量的檢驗，將消除資源衝突或可能帶來新的資源衝突的狀況，立刻反應給設計者知道。這些 activity processes 的異動因素，包含了插入新增、刪除 process、改變 process 內的資源參考，接著再加入時間因素考量，分析當一個 process 的最小工作時限與最大工作時限有改變時，所造成的影響程度以及必須檢驗的範圍。

本文的研究重心擺在設計階段透過探討互動式分析 process 的改變與影響，最後利用分支路徑資源串列記錄表與 EAI 時間表，達成遞增式分析且加速了分析工作的效能。未來研究將朝向：(1)複雜的不同資源類別探討，(2)在執行階段動態改變工作流程的規格定義，如何針對資源衝突檢驗提供動態分析，(3)本文提及的遞增式資源檢驗分析，如何與目前已知的工作流程規格設計工具互相作整合、改善，也是未來實作時所考慮的重要項目。

文獻參考

[1] WMC, Workflow Management Coalition. http://www.wfmc.org/

[2] Hongchen Li, Yun Yang, T.Y. Chen, “Resource constraints analysis of workflow specifications”, the Journal of System and Software 73 (2004).

[3] Sadiq, W., Orlowska, M.E., “Analysing process models using graph reduction techniques”, Information System 25(2), p117-134. 2000.

[4] Marjanovic, O., “Dynamic verification of temporal constraints in production workflows.”, Proceedings of the Australian Database Conference. IEEE Press, Canberra, Austalia, pp. 74-81.

[5] Peter J. Kammer and David W. McDonald, “Putting Words to Work: Integrating Conversation with Workflow Modeling”, Technical Report 99-30.

[6] Aalst, W.M.P.v.d., Basten, T., Verbeek, H.M.W., Verkoulen, P.A.C., and Voorhoeve., M. “Adaptive Workflow: An Approach Based on Inheritance.” In Proceedings of the IJCAI’99 Workshop on Intelligent Workflow and Process Management: The New Frontier for AI in Business. 1999. Stockholm, Sweden.

[7] Fleurke, M and Ehrler, L, Purvis, M. A. (2003). “JBees - An Adaptive and Distributed Agent-based Workflow System”, in Proceedings of the International Workshop on Collaboration Agents: Autonomous Agents for Collaborative Environments (COLA 2003), Halifax, Canada, October 2003.

[8] Michael zur Muehlen, Rob Allen, “Workflow Classification: Embedded and autonomous Workflow”. WfMC White Paper. Lighthouse Point, March 10, 2000.

[9] Rob Allen, “Workflow: An Introduction”, Workflow Handbook 2001.

[10] Adam, N., Atluri, V., Huang W., 1998. “Modeling and Analysis of Workflows using Petri-Nets”, Journal of Intelligent Information System, 10(2), 1998.

[11] M. Reichert, P. dadam. “ADEPT-Supporting Dynamic Changes of Workflows

without Losing Control", Journal of Intelligent Information System, 10(2), 1998.

[12] D. Widtke, G. Weikum. "A Formal Foundation for Distributed Workflow Execution based on State Charts", Proceedings of the International Conference on Database Theory, Springer Verlag LNCS 1186, 1997.

[13] Onoda, S., Ikkai, Y., Kobayashi, T., Komoda, N.,1999. "Definition of deadlock patterns for business processes workflow models." Proceedings of the 32nd Hawaii International Conference on System Sciences, pp. 1-11.

[14] Singh, M.P., "Formal aspects of workflow management, Part 1: Semantics." Technical Report, Department of Computer Science, North Carolina State University, 1997.

[15] J. Rakos. "Software Project Management for small to medium sized Projects", Prentice Hall, 1990.

[16] Eder, J., Panagos, E., Rabinovich, M., 1999. "Time Constraints in Workflow Systems." Proceedings of the 11th International Conference on Advacned Information Systems Engineering. (CAISE 1999). Springer Verlag, LNCS 1626, Germany, pp. 286-300.

[17] Zhunge, H., Cheung, T., Pung, H., 2001. "A Timed Workflow Process Model." The journal of Systems and Software 55(3), 231-243.

[18] Hans van Vliet, "Softeare Engineering: Principles and Practice", 2nd edition by John Wiley & Sons, Ltd. 2000.

[19] Shazia Sadiq, Maria Orlowska, Wasim Sadiq, Cameron Foulger, "Data Flow and Validation in Workflow Modeling", Conferences in Research and Practice in Information Technology, Vol. 27. 2003.

[20] Flowring Corp., AgentFlow system, http://www.flowring.com

[21] Hofstede A., Orlowska, M., E, 1999. "On the complexity of some verification problems in process control specifications", the Computer Journal 42(5)

pp349-359.

[22] Tarumi, H., Matsuyama, T., Kamabayashi, Y., 1999. “Evolution of business processes and a process simulation tool.” In: Proceedings of Asia Pacific Software Engineering Conference (APSEC ’99), pp.180-187.

[23] Kim, K., Ellis, C.A., 2001. “Performance analytic models and analyses for workflow architectures.” Journal of Information Systems Frontiers 3(3), pp.339-355.

[24] C.Capellmann, H. Dibold: “Petri Net Based Specifications of services in an Intelligent Network”, “Experients Gained from a Test Case Application.” In: M. Ajmone-Marsan (ed.): Applicationn and Theory of Petri Nets 1993. Proceedings of the 14th International Petri Net Conference, Chicago 1993, Lecture Notes in computer Science Vol. 691, Springer-Verlag 1993, pp.542-551.

[25] G. Balbo, S.C. Bruell, P. Chen, G. Chiola: An Example of Modleing and Evaluation of a Concurrent Program Using Colored Stochastic Petri Nets: Lamport’s Fast Mutual Exclusion Algorithm. IEEE Transactions on Parallel and Distributed System, 3(1992). Also in [27], pp.533-559.

[26] J. Berger, L. Lamontagne, "A Colored Petri Net Model for a Naval Command and Control System", in M. Ajmone-Marsan (ed.): Application and Theory of Petri Nets 1993. Proceedings of the 14th International Petri Net Conference, Chigago 1993, Lecture Notes in Computer Science Vol. 691, Springer-Verlag 1993, pp.532-541.

[27] K. Jensen, G. Rozenberg (eds): "High-level Petri Nets. Theory and Application." Springer-Verlag, 1991.

[28] David Hollingsworth, “The Workflow Reference Model”, 1995

[29] David Hollingsworth, “The Workflow Reference Model: 10 Years On”, 2004