1-1-2014

Explicit Preemption Placement For Real-Time Conditional Code Via Graph Grammars And Dynamic Programming

Bo Peng
Wayne State University,

EXPLICIT PREEMPTION PLACEMENT FOR REAL-TIME CONDITIONAL CODE VIA GRAPH GRAMMARS AND DYNAMIC PROGRAMMING

by

BO PENG

THESIS

Submitted to the Graduate School

of Wayne State University,

Detroit, Michigan

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

2014

MAJOR: COMPUTER SCIENCE

Approved by:

_____

Advisor Date

DEDICATION

To my Father and Mother.

ACKNOWLEDGMENTS

I would like to express my deeply gratitude to my advisor, Dr. Nathan Fisher. He gives me patience, encouragement, and writing techniques to help me finish this thesis. He offers me good advice and recommends good books on research. His explanations to my questions are quite detailed and logical. I'm fortunate and honored to work with Dr. Nathan Fisher. I'm also grateful to Dr. Loren Schwiebert and Dr. Marwan Abi-Antoun for spending time reading this thesis and giving good comments.

The research presented in this thesis has been funded by the US National Science Foundation (Grant no. CNS-0953585). Any opinions, findings, and conclusions or recommendations expressed in this thesis are those of the author and do not necessarily reflect the views of the National Science Foundation.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Real-time and embedded systems, which span a broad scope of complexity from micro-controllers to highly complicated and distributed systems, require completion of computation and delivery of service in a timely fashion. The correctness of an operation depends not only on its logical correctness, but also on the time in which the operation is performed. Such systems, subject to the time constraints, must guarantee that successful completion of execution does not exceed its deadline. Thus, in order to obtain deadline guarantees, the running time of each operation must be carefully calculated. An upper bound for running timing requirement of each operation is needed. The determination of the upper bounds on execution times depends on various input data and different behavior of the processing environment.

In real-time Worst-Case Execution Time (WCET) analysis, an upper bound is calculated, for each job in the system, on the total aggregate amount of execution required to successfully complete the job. Real-time schedulability analysis has traditionally used the estimates derived from a WCET analysis to determine whether every job in a system can be completed by its deadline. Thus, the effectiveness of the resulting schedulability analysis hinges upon the precision of WCET estimates. Unfortunately, many scheduler properties that simplify schedulability analysis often introduce pessimism into WCET analysis. For example, the oft-assumed property that jobs are arbitrarily preemptible leads to a significant increase in WCET estimates, as the analysis must assume that a preemption occurs often and the overhead of such preemption (due to context switch time and cache effects) must be added to the estimate.

Furthermore, most real-time scheduling algorithms and associated schedulability analysis do not take the heterogeneous "cost" of preemption overhead into account when

making scheduling decisions. For instance, preemption of a job may cause cache lines that are needed in subsequent instructions to be invalidated; the memory access pattern of nearby instructions will greatly influence the cost of the preemption due to such invalidations. Thus, a better strategy may be to delay the preemption of a job that is executing instructions with a degree of spatial or temporal locality (in terms of memory access) until it reaches instructions with a lower level of memory locality.

1.1 Thesis and Contribution

Our thesis is as follows:

The optimal placement of explicit preemption point, which is decided in the code containing conditional operations, can be solved in pseudo-polynomial time in the number of basic blocks and the duration of the maximum non-preemptive region. The final result is a fully integrated approach that allows the automatic selection of preemption points for applications that can be modeled as a flow of basic block code.

Bertogna et al. [6] proposed an approach that explicitly and efficiently determines (prior to runtime) the optimal choice of *Explicit Preemption Points* (EPPs) in a job's code that minimize the preemption overhead while ensuring that system schedulability is not affected due to increased non-preemptivity. However, their proposed approach only deals with linear (non-branching) code and cannot handle jobs with control flow such as conditional statements (e.g., if-then-else statements) and loops.

1.2 Organization

In Chapter 2, we introduce the notations and formally state the explicit preemption placement problem. Related work is discussed in Chapter 3. Chapter 4 gives an overview of *graph grammars*, a useful approach to recognizing and processing programs based on their control flowgraphs. Chapter 5 is the core part of this thesis; after giving an example that shows why the linear method does not work, we propose a dynamic-programming-

based approach to obtain an optimal EPP placement of a given flowgraph. We first give a recursive formulation of the EPP placement problem by exploiting the structure defined by the graph grammar for conditional real-time control flowgraphs. Then, we show a high-level intuitive overview of the algorithm. Last, we focus on providing more details on the implementation of our proposed algorithm. Chapter 6 evaluates the performance of the proposed preemption point placement methods over different kinds of randomly-generated control flowgraphs by comparing it to straightforward extensions of the linear method for conditional code. Conclusions and future works are presented in Chapter 6.

CHAPTER 2

MODEL AND PROBLEM STATEMENT

Typically, a system consists of a central processing unit and some form of memory. Each unit of work that is scheduled and executed by the system is defined as a *job*. A set of related jobs that jointly provide some function form a *task*. A task system τ consists of n sporadic tasks [18] that are scheduled on processors. A job $j = (re, ex, de)$ is characterized by three parameters – a release time re, an execution requirement ex, and a deadline de – with the interpretation that the job needs to be executed for an amount equal to its execution requirement between its release time and its deadline. A sporadic task T_i can be characterized by a three-tuple (ex_i, de_i, pe_i). The execution time ex_i denotes an upper bound execution requirement for each job. The relative deadline de_i denotes the time range between each job's arrival time and deadline. The period pe_i denotes the separation between the arrival times of successive jobs.

The execution of a task is represented by a workflow consist of a set of non-preemptive Basic Blocks (BBs) which contain a few jobs each. Potential preemptions may occur between any two consecutive BBs. We select the Effective Preemption Points (EPPs) from the Potential preemption points (PPPs) to get the optimal (smallest) preemption overhead. The sequence of basic blocks between any two consecutive EPPs forms a Non-Preemptive Region (NPR). Critical Sections are assumed to be executed within a BB; thus, we need not use shared resource protocols to access critical sections.

As stated above, we refer to the problem of determining the optimal choice of EPPs for a program (i.e., job) as the *explicit preemption placement* problem. Let $V = \{\delta_1, \delta_2, \ldots, \delta_n\}$ be the set of basic blocks. A *control flowgraph* $G_{\mathcal{P}} = (V, E, \delta_s, \delta_z)$ describes the control flow of $\mathcal{P}$ (program). V is the set of basic blocks; E is the set of edges. A flowgraph is a directed graph (V, E) with a distinguished start vertex (basic block) δ_s and an exit

vertex (basic block) δ_z such that for $\delta_v \in V$ there is a path from δ_s to δ_v and a path from δ_v to δ_z. Clearly, each vertex $\delta_v \in V$ represents a BB in $\mathcal{P}$. An edge $(\delta_u, \delta_v) \in E \subseteq V \times V$ means that the execution of BB δ_u immediately precedes the execution of BB δ_v in some execution path of $\mathcal{P}$, and that a preemption is permitted between the two BBs, i.e., E is the set of possible EPPs. A path p is an ordered set of consecutive vertices, such that each vertex in p has an edge from its predecessor. Let $\mathsf{paths}(G_{\mathcal{P}}, \delta_x, \delta_y)$ be the set of possible execution paths between the basic blocks δ_x and δ_y in the control flowgraph $G_{\mathcal{P}}$. We say that $\delta_u \preceq_p \delta_v$, if δ_u precedes (or equals) δ_v along path $p \in \mathsf{paths}(G_{\mathcal{P}}, \delta_x, \delta_y)$. The operator $\prec_p$ denotes strict precedence.

For the purposes of quantifying the preemption overhead of selecting an EPP, we assume that a CRPD function $\xi : E \mapsto \mathbb{R}_{\geq 0}$ is given. If preemption is not permitted between basic blocks $\delta_u, \delta_v \in V$ for edge $(\delta_u, \delta_v) \in E$, we can model this scenario by setting $\xi(\delta_u, \delta_v)$ equal to ∞. Similarly, the WCET of a BB is given by a function $C : V \mapsto \mathbb{R}_{\geq 0}$. We assume that there are two "dummy" *sentinel basic blocks* $\delta_{-\infty}$ and δ_∞ such that $(\delta_{-\infty}, \delta_s)$ and $(\delta_z, \delta_\infty)$ are in E and $C(\delta_{-\infty}) = C(\delta_\infty) = \xi(\delta_{-\infty}, \delta_s) = \xi(\delta_z, \delta_\infty) = 0$. Edges $(\delta_{-\infty}, \delta_s)$ and $(\delta_z, \delta_\infty)$ are called *sentinel edges*. Figure 2 shows a corresponding example of flowgraph.

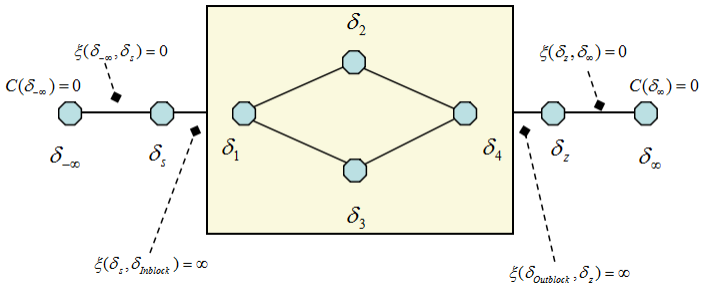


Figure 2.1: An example of a control flowgraph.

2.1 Series-parallel graphs

For this thesis, we will study an important class of control flowgraphs called *series-parallel graphs* [26]. That is, $G_{\mathcal{P}}$ can be obtained by applying some sequence of the following three operations:

1. Graph Creation: create a graph with two nodes and a single directed edge between them.
2. Series Composition: given two series-parallel graphs $G_{\mathcal{X}}$ and $G_{\mathcal{Y}}$, a new graph is created by merging the sink node of $G_{\mathcal{X}}$ with the source node of $G_{\mathcal{Y}}$.
3. Parallel Composition: given two series-parallel graphs $G_{\mathcal{X}}$ and $G_{\mathcal{Y}}$, a new graph is created by merging (i) the source nodes of $G_{\mathcal{X}}$ and $G_{\mathcal{Y}}$, and (ii) the sink nodes of $G_{\mathcal{X}}$ and $G_{\mathcal{Y}}$.

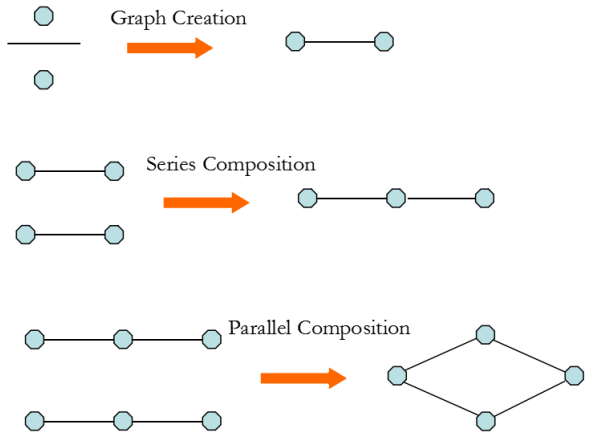


Figure 2.2: An example of a series-parallel graph.

Note that previous work on explicit preemption placement [6] handled only graphs created using the series composition operation.

Series-parallel graphs are appropriate to model many structured programming language constructs such as `if-then-else` statements, `switch` statements, and loops with bounded iterations [2] where programs can easily be subdivided into segments with a single entry point and a single exit point. Wilhelm et al. [25] states that: "real-time

systems only use a restricted form of programming, which guarantees that programs always terminate, recursion is not allowed or explicitly bounded, as are the iteration counts of loops". Therefore, the adopted task model based on series-parallel graphs can be efficiently used to model real-time tasks implemented via a structured programming language. Note that loops may be modeled as series compositions of multiple blocks. A long loop can then be simply split into the series composition of different sub-loops, each one mapped to a basic block of smaller granularity. The size of such sub-loops can be freely decided at design time.

The only limitation of the series-parallel graph model is in the preclusion of `goto` statements and early returns. However, Böhm and Jacopini [8] proved that every structured program can be expressed as a combination of sequential instructions, conditional branches and loops, without needing `goto`'s and multiple exit points.

2.2 Problem Statement

Let $\mathcal{G}$ be the set of flowgraphs that define programs with conditional control structures according to the above model. Since the selection of EPPs will create non-preemptible regions in $\mathcal{P}$, the schedulability of the system is affected by a choice of EPPs. We will assume that a constant Q is determined which quantifies the maximum duration of any non-preemptive region in $\mathcal{P}$. This parameter depends on the task set characteristics and on the adopted scheduling algorithm. Methods to compute Q for EDF and Fixed Priority scheduled systems are presented in [4] and [7], respectively.

Given the above model, our goal is to find a selection of EPPs that minimizes the WCET+CRPD of $\mathcal{P}$, without imposing non-preemptive regions that are greater than the maximum allowed non-preemptive execution Q. More formally, the main problem addressed in this paper is the following.

Problem statement:

Given $G_{\mathcal{P}} \in \mathcal{G}$ *and associated functions* ξ *and* C*, find* $S \subseteq E$ *that minimizes*

$$\Phi(G_{\mathcal{P}}, S) \stackrel{def}{=} \max_{p \in \mathsf{paths}(G_{\mathcal{P}}, \delta_s, \delta_z)} \left\{ \sum_{\delta_u \in p} C(\delta_u) + \sum_{\substack{\delta_u, \delta_v \in p \\ (\delta_u, \delta_v) \in S}} \xi(\delta_u, \delta_v) \right\} \tag{2.1}$$

subject to the constraint that $\forall p \in \mathsf{paths}(G_{\mathcal{P}}, \delta_s, \delta_z), \delta_i \in p$*:* $\exists e_1 = (\delta_u, \delta_v), e_2 = (\delta_x, \delta_y) \in S$ ::

$$(\delta_u \preceq_p \delta_i \preceq_p \delta_y) \quad \wedge \quad \left(\xi(e_1) + \sum_{\substack{\delta_j \in p \\ \delta_v \preceq_p \delta_j \preceq_p \delta_x}} C(\delta_j) \leq Q \right). \tag{2.2}$$

In words, the last constraint means that for each path p in paths and for any basic block $\delta_i \in p$, there exist two EPPs in S such that the non-preemptive region between such EPPs contains δ_i and has a total execution cost no larger than the schedulability constraint Q. If no such S exists, we say that $G_{\mathcal{P}}$ is not feasible for the given Q.
In the above problem statement, the Φ function corresponds to the WCET+CRPD of $\mathcal{P}$ when we have the non-preemptivity constraint of Q.

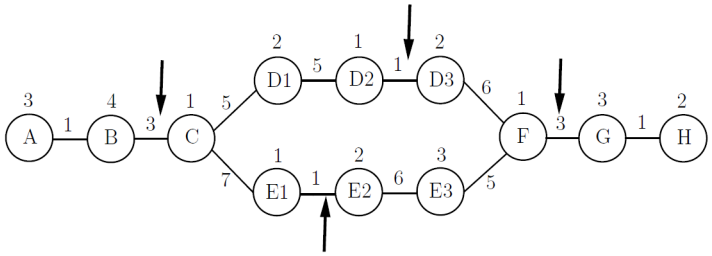


Figure 2.3: Example of optimal EPP selection.

Figure 2.3 shows a simple control structure including a conditional branch. By a brute-force approach, we find the optimal selection of EPPs shown as arrows in the figure. There are two paths in the control structure. The $\sum_{\delta_u \in p} C(\delta_u)$ of upper path is the sum of basic block costs, which is $C(\delta_A) + C(\delta_B) + C(\delta_C) + C(\delta_{D1}) + C(\delta_{D2}) +$

$C(\delta_{D3}) + C(\delta_F) + C(\delta_G) + C(\delta_H) = 19$. The $\sum_{\substack{\delta_u, \delta_v \in p \\ (\delta_u, \delta_v) \in S}} \xi(\delta_u, \delta_v)$ is the overhead of EPPs, which is $\xi(\delta_B, \delta_C) + \xi(\delta_{D2}, \delta_{D3}) + \xi(\delta_F, \delta_G) = 7$. The WCET+CRPD of the upper path is $19 + 7 = 26$. The constraint states that the maximum NPR should be less than or equal to Q. Similarly, the WCET+CRPD of the lower path is 27. Then, we choose the 27 as the bound on the worst-case execution times of this limited preemptive task. Please notice that, for a single path considered in isolation, the selection of EPPs might not be optimal for that specific path; however, a selection of EPPs that are not optimal for each individual path can still be globally optimal overall paths if they minimize the WCET+CRPD. The details of how to minimize the maximum WCET+CRPD of $G_{\mathcal{P}}$ are presented in Chapter 5.

One note about solutions for the above problem: we seek *optimal* solutions under the assumption that a preemption will occur at each EPP (i.e., the worst case will occur). Clearly, if the system scheduler is lightly loaded and can tolerate non-preemptively executing a task for more than Q time units, some other selection of preemption points may result in lower overhead at runtime. However, since we are interested in knowing at design-time the WCET+CRPD and minimizing the contribution of preemptions to the WCET+CRPD, this notion of optimality seems the most natural and appropriate.

CHAPTER 3

RELATED WORK

Preemptive scheduling model provides a flexible frame compared to the models with non-preemptive scheduling algorithms [20]. The flexible frame reflects the fact that a operating system stop the running service then allocate the processor to the incoming urgent service. However, papers ([11], [14], [17], [19], [23], [21]) have shown that arbitrary preemption cause complicated system design and analysis. Grenier and Navet [15] states that preemption in I/O scheduling is prohibitively expensive or even impossible. Limited preemptive algorithms arise as alternative scheduling schemes between preemptive scheduling algorithms and non-preemptive scheduling algorithms.

The research on limited preemptive scheduling algorithms has recently received significant attention due to performance benefits in terms of reduced preemption overhead and increased predictability. Wang and Saksena [24] proposed a approach that the task won't be preempt until the incoming task's priority is larger than the priority threshold of the running task. According to this scheduling model, each task is divided into a set of non-preemptive regions so that preemptions can take place only at a subset of points. Depending on the location of such non-preemptive regions, limited preemptive schedulers are divided into two sub-categories: fixed and floating preemption point models.

The fixed preemption point model has been introduced by Burns [10]. According to this model, tasks are divided into statically defined non-preemptive chunks of fixed length, so that preemptions are allowed only at chunk's boundaries. Since tasks cooperate in order to offer suitable preemption points to decrease the context switch overhead, such a model is also called Cooperative Scheduling.

A tight schedulability analysis for the fixed preemption point model has been presented by Bril et al. [9].

In the floating preemption point model, instead, the size and location of the non-preemptive regions are not known before run-time, but are determined during task execution. Only an upper bound is given on the maximum allowed non-preemptive region Q of a task. The floating model has been adopted by Baruah et al. [3,4] for EDF scheduled systems, and by Yao et al. [27] for Fixed Priority scheduled systems. In both works, upper bounds are provided on the largest non-preemptive region Q that can be enforced in each task without causing any deadline miss.

The problem of computing the optimal Q in the fixed preemption point model has been addressed by Bertogna et al. [7] for fixed priority systems, showing that inserting a non-preemptive region at the end of a task might increase the schedulability of the system with respect to fully preemptive or non-preemptive scheduling. In the same paper, an algorithm is shown to compute the largest non-preemptive region Q for each task in order to maximize the schedulability of the system. Davis et al. later adapted this method [13], showing an optimal priority assignment to be used in combination with the fixed preemption point model.

When preemption overhead is included in the analysis, the derived bounds on the largest non-preemptive region Q of each task can be exploited to reduce the preemption overhead as much as possible without compromising feasibility. The fixed preemption point model can be adopted to insert preemption points at suitable locations, such that the length of each non-preemptive region does not exceed Q. Bertogna et al. [5] presented an optimal preemption point placement method with linear complexity under the assumption that the preemption overhead is constant throughout the code of each task. Bertogna et al. [6] have relaxed this assumption, considering a variable preemption overhead and selecting the optimal subset of preemption points that maximizes the schedulability of the task system. The preemption point placement algorithm has pseudo-polynomial complexity, proportional to Q and to the number of potential pre-

emption points. In both papers, a linear task structure is considered, so that each task is composed of a linear sequence of basic blocks, and if-then-else constructs are entirely contained inside a BB. An survey [12] is available on comparing different limited preemptive scheduling techniques.

CHAPTER 4

FLOWGRAPHS FOR REAL-TIME CONDITIONAL CODE

The definition of control flowgraph is broad enough to permit a wide variety of programmatic structures. In this paper, we are focused on analyzing programs that have conditional control structures such as `if-then-else` or `switch` constructs. To be precise, we need to be specific about the types of control flowgraphs that we will address in this paper. Typically, a programming language designer defines a *context-free grammar* to specify the set of strings that are considered valid programs in the programming language. However, in this paper, our input is a control flowgraph $G_{\mathcal{P}}$; thus, we need to specify the subset of control flowgraphs for which our approach applies. For this purpose, we create a *flowgraph grammar* [16] over control flowgraphs to specify the set of graphs (i.e., programs) that we consider valid conditional real-time control flowgraphs. In this chapter, we first give a brief background on graph grammars in Chapter 4.1. Then, we give the formal specification of the graph grammar for the real-time conditional code considered in Chapter 4.2.

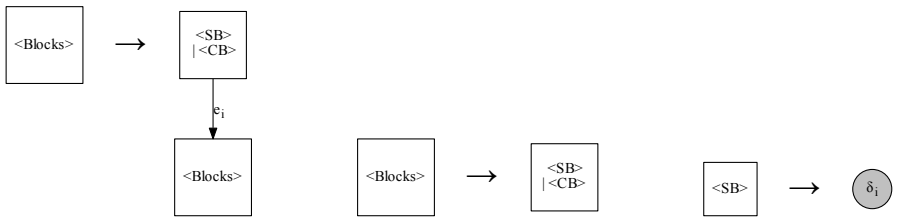
4.1 Graph Grammars

In the late 1970s and early 1980s, an effort was made in the programming languages community to understand the structure of control flowgraphs resulting from program structure. The concept of *graph grammars* was introduced as an attempt to formalize the identification of different programmatic structures and develop analysis tools for the purpose of code optimization [16].

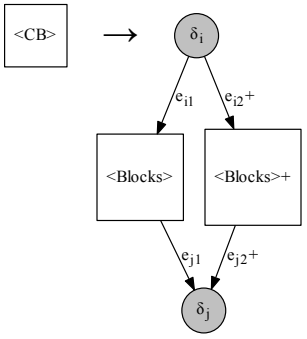
In this thesis, we utilize the concept of graph grammars for two reasons: 1) graph grammars provide a formalism for automatically describing and recognizing the set of

control flowgraphs that satisfy our task model; and 2) they permit an elegant means of expressing an algorithm to solve the EPP problem for real-time conditional code.

A graph grammar is the means of specifying the "syntax" of proper control flowgraphs. Like a textual program, a control flowgraph contains tokens; however, instead tokens being strings of number, letters, or symbols, as in a textual program, the set of tokens in a control flowgraphs are vertices and edges. The graph grammar is the set of rules specifying how these "tokens" may be combined to form a valid control flowgraph. Each rule in a graph grammar is called a *production* (or also referred to as a *graph-rewriting rule* in the graph grammar literature). A production rule has a left-hand side and a right-hand side. The left-hand side of a production contains a *non-terminal node*; the non-terminal node is an abstract representation of some collection of vertices and edges in the control flowgraph. The right-hand side of the production contains non-terminal node(s) and/or *terminal node(s)*. The collection of these nodes may be connected by edges also specified in the right-hand side of the production rule. Each terminal node is a vertex in the control flowgraph and, in our setting, it represents a BB. The production specifies that the non-terminal node on the left-hand side of the production may be substituted (or rewritten) with the nodes on the right-hand side of the production; this process of substituting is called a *derivation*. A graph grammar $\mathcal{G}$ is a set of production rules which defines a specific graph language $L(\mathcal{G})$ generated by graph grammar $\mathcal{G}$. A graph G is in the language $L(\mathcal{G})$, if there exists a sequence of derivations, starting from a specified starting non-terminal node, that uses productions of $\mathcal{G}$ and results in graph G. A graph grammar is called *context-free* if each production has only non-terminal nodes on the left-hand side of the production. A graph grammar is called *unambiguous* if for each $G \in L(\mathcal{G})$ there exists a unique sequence of derivations for G. In the next subchapter, we give the production rules for our control flowgraph grammar of real-time conditional code.



(a) $\langle \texttt{Blocks} \rangle \rightarrow (\langle \texttt{SB} \rangle | \langle \texttt{CB} \rangle), e_i, \langle \texttt{Blocks} \rangle$ (b) $\langle \texttt{Blocks} \rangle \rightarrow (\langle \texttt{SB} \rangle | \langle \texttt{CB} \rangle)$ (c) $\langle \texttt{SB} \rangle \rightarrow \delta_i$



(d) $\langle \texttt{CB} \rangle \rightarrow \delta_i, \left[e_{i_1}, \langle \texttt{Blocks} \rangle, e_{j_1}\right], \left(e_{i_2}, \langle \texttt{Blocks} \rangle, e_{j_2}\right)+, \delta_j$

Figure 4.1: Production rules for control flowgraph grammar $\mathcal{G}$.

4.1.1 High-Level Overview of Approach

The basic idea behind our approach is that we can compute the optimal EPPs for subgraphs in the program's flowgraph and utilize the subgraph solutions to determine the overall optimal EPP solution for the entire flowgraph. At a high level, we will begin by determining the EPP for innermost conditional blocks first and then work our way to the outermost flowgraph structures. For instance, in the example flowgraph in Figures 4.2 and 5.1, the optimal choice of EPPs for the conditional block beginning at basic block C and ending at basic block F will be determined prior to calculating the EPPs for the overall flowgraph. Unfortunately, it is not sufficient to calculate a single EPP solution for a substructure due to the dependence on EPP placement with EPP selected by the larger structure; for instance, the optimal choice of EPPs for the conditional block in

Figure 4.2 depends upon the last preemption that occurs before basic block C (e.g., is (A, B) or (B, C) the last preemption before C?) and the first preemption selected after basic block F (e.g., is (F, G) or (G, H) the first preemption after F?). The reason for this dependence is that the choice of preemptions within the nested structure must satisfy the constraint in the larger structure that EPPs are no further than Q units apart (i.e., the constraint of Equation 2.2).

A natural question at this point is: *how do we compute the optimal EPPs for inner substructures first when the preemption placement is dependent upon the EPP selection in outermost structures?* The answer is to compute and store the optimal EPP selection/cost for all possible preemption values before and after each substructure. In other words, assuming for any substructure $G^A_{\mathcal{P}}$ of the input flowgraph that (i) the last preemption before substructure $G^A_{\mathcal{P}}$ occurs ζ_1 time before the first basic block of $G^A_{\mathcal{P}}$, and (ii) the earliest preemption after $G^A_{\mathcal{P}}$ occurs ζ_2 time units after the last basic block of $G^A_{\mathcal{P}}$. Then, a set $S^A(\zeta_1, \zeta_2)$ and cost-matrix $\mathsf{cost}(G^A_{\mathcal{P}}, \zeta_1, \zeta_2)$ will be computed for all possible values of ζ_1 and ζ_2. The set $S^A(\zeta_1, \zeta_2)$ represents the optimal EPP selection (w.r.t. Equations 2.1 and 2.2) for substructure $G^A_{\mathcal{P}}$ given the preemptions' times before and after $G^A_{\mathcal{P}}$ are ζ_1 and ζ_2. The cost-matrix $\mathsf{cost}(G^A_{\mathcal{P}}, \zeta_1, \zeta_2)$ represents the corresponding WCET+CRPD cost for substructure $G^A_{\mathcal{P}}$ with EPP selection $S^A(\zeta_1, \zeta_2)$. For instance, in the example of Figure 5.1, when $G^A_{\mathcal{P}}$ represents the conditional block staring at C and ending at F, the set $S^A(3, 3)$ equals $\{(D2, D3), (E1, E2)\}$ and $\mathsf{cost}(G^A_{\mathcal{P}}, 3, 3)$ equals 8.

Given the above approach, the reader may also wonder: *how many subproblems for each substructure need to be solved?* By the constraint of Equation 2.2, the largest value for preemptions before or after any substructure (i.e., upper bounds for values of ζ_1 and ζ_2) is Q time units. Furthermore, if we assume that due to the clock tick granularity of the system preemption times must be integers, then there are at most $Q \times Q = Q^2$ combinations of ζ_1 and ζ_2; thus, there are at most Q^2 subproblems we must

compute for each substructure. In the next subsection, we obtain a recursive formulation which computes the optimal EPP selection for any substructure based on the optimal solutions to the subsubproblems. This is a classic dynamic programming approach. After obtaining the recursive formulation, Subchapter 5.2 will describe a more efficient bottom-up implementation.

4.2 Real-time Conditional Flowgraph Grammar Specification

Figure 4.1 gives the production rules for graph grammar $\mathcal{G}$ for control flowgraphs that satisfy our task model. A boxed node represents a non-terminal node; a circle node represents a basic block (i.e., a terminal node). We can also represent the graph grammar by a traditional text-based grammar. The text-based equivalent production is given along with the graph grammar production. We use *Extended Backus-Naur Form* (EBNF) to describe both the graph grammar $\mathcal{G}$ and its equivalent text-based representation. A term in angle brackets (e.g., $\langle$`Blocks`$\rangle$) indicates that this is a non-terminal node. The $x \mid y$ operator means that either the term x or y may be used in the production. The expression $x+$ indicates that one or more successive occurrences of the expression x may be used in the substitution. Any programming languages textbook will contain more complete and formal descriptions of EBNF and the terminology of Chapter 4.1 (as it applies to textual grammars). For a good textbook on programming languages and their grammars, we refer the reader to Scott [22].

Figure 4.1(a) shows that a set of blocks (represented by $\langle$`Blocks`$\rangle$) consists of either a single block ($\langle$`SB`$\rangle$) or a conditional block ($\langle$`CB`$\rangle$), connected by edge e_i to another set of blocks. Figure 4.1(b) indicates that a set of blocks could just comprise a single sequential or conditional block without any subsequent blocks. Figure 4.1(c) shows that a single block is a simple BB δ_i. Finally, Figure 4.1(d) gives the production for conditional blocks:

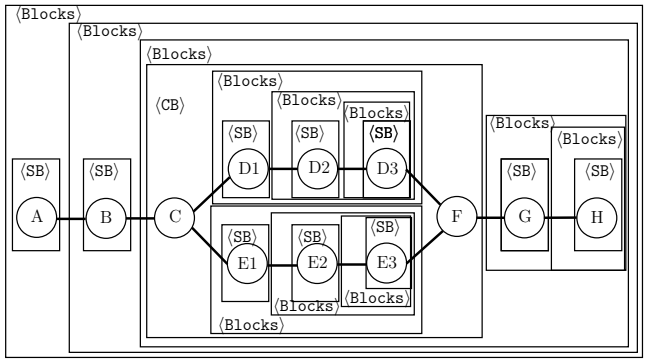


Figure 4.2: A derivation using the production rules of Figure 4.1 over an example control flowgraph.

a conditional block has a forking BB δ_i with two or more outgoing edges and a joining BB δ_j with an equivalent number of incoming edges. If there are k paths from δ_i to δ_j in the conditional block, then for any path p_ℓ $(1 \leq \ell \leq k)$ a collection of blocks is contained on the path between δ_i and δ_j and connected to these BBs by edges e_{i_ℓ} and e_{j_ℓ}, respectively. Clearly, this grammar permits multiple nested levels of conditional blocks, as the blocks within each path may consist of additional conditional blocks. Note that a conditional block with two paths can represent an `if-then-else` construct in a program; a conditional block with two or more paths can represent a `switch` construct. It can easily be seen that $\mathcal{G}$ represents an unambiguous context-free grammar. Figure 4.2 shows the application of the production rules given in Figure 4.1.

CHAPTER 5

DYNAMIC-PROGRAMMING ALGORITHM

As mentioned in Chapter 3, an optimal preemption point placement method has been presented in [6] for linear task structures, i.e., without any conditional block. One might wonder whether an optimal solution can be obtained by applying the method in [6] to the worst-case path of a conditional task structure. Or, alternatively, by repeatedly applying it to each possible path in a conditional task structure. Unfortunately, this is not the case, since the "globally" optimal solution (i.e., the EPP placement resulting in the smallest possible WCET+CRPD) might differ from the combination of the optimal solutions of each path. Consider the example in Figure 5.1, where a simple task structure including a conditional branch is depicted. The WCET of each BB and the preemption cost at each edge are specified above each node and each edge, respectively. The maximum allowed non-preemptive section is assumed to be $Q = 8$.

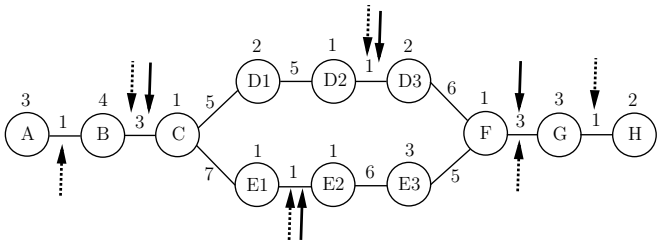


Figure 5.1: Example of EPP selection using the linear method in [6] for each path (dashed arrows) w.r.t. the optimal EPP selection (solid arrows).

If the linear method in [6] is applied to the upper branch, three EPPs are placed: one between nodes B and C, one between $D2$ and $D3$, and the last one between G and H. When the same method is applied to the lower branch, one EPP is placed between nodes A and B, one between $E1$ and $E2$, and another one between F and G. Altogether, six different EPPs are placed in the graph, leading to an overall preemption overhead of 9 time-units in each of the paths, and a total WCET+CRPD of 28 time-units. However, a

smarter placement of EPPs is able to reach a smaller WCET+CRPD by taking a "global" view of the task structure. A WCET+CRPD of 26 can be obtained selecting four EPPs: one between nodes B and C, one between $D2$ and $D3$, one between $E1$ and $E2$, and the last one between F and G, with a total preemption cost of 8 time-units in each of the paths. Note that the selected placement does not coincide with any of the two linear selections.

In this chapter, we describe a dynamic-programming-based approach for obtaining an optimal solution to the EPP placement problem. Our returned solution $S^* \subseteq E$ is optimal in the sense that any other $S' \subseteq E$ must result in a higher or equal WCET+CRPD for $\mathcal{P}$; in other words, $\Phi(G_{\mathcal{P}}, S^*) \leq \Phi(G_{\mathcal{P}}, S')$ where Φ is the objective function defined in Equation (2.1) of our problem statement. In the previous chapter, each non-terminal node (i.e., $\langle$`SB`$\rangle$, $\langle$`CB`$\rangle$, and $\langle$`Blocks`$\rangle$ consists of a collection of basic blocks and other non-terminal nodes. Thus, there is a well-defined structure that we may potentially exploit to determine the optimal placement of EPPs within program $\mathcal{P}$.

In the next Chapter 5.1, we give a recursive formulation of the EPP placement problem by exploiting the structure defined (in Chapter 4) by the graph grammar for conditional real-time control flowgraphs.

5.1 Optimal Substructure & Recursive Formulation

A derivation of $\mathcal{G}$ on an input graph $G_{\mathcal{P}}$ is a sequence of production rules applied to the input graph. During the derivation, the underlying basic blocks are assigned to the non-terminal nodes (see Figure 4.2 for an example). Let A be any production in the derivation for $G_{\mathcal{P}}$. Let $G^A_{\mathcal{P}}$ denote the subgraph induced by production A in $G_{\mathcal{P}}$ derivation over grammar $\mathcal{G}$. In other words, $G^A_{\mathcal{P}} = (V^A, E^A)$ is the collection of basic blocks and edges represented by the non-terminal block on the left-hand side of production A.

The graphs that can be obtained using the productions rules of grammar $\mathcal{G}$ specified in Chapter 4.2 are also called "series-parallel graphs". For this kind of graph, it may be

shown [26] that for any production A, there exists for $G^A_{\mathcal{P}}$ exactly one node (i.e., basic block) with in-degree equal to zero and exactly one node with out-degree equal to zero. Let $\mathsf{InBlock}(G^A_{\mathcal{P}})$ denote the node of $G^A_{\mathcal{P}}$ with in-degree equal to zero and $\mathsf{OutBlock}(G^A_{\mathcal{P}})$ as the node with out-degree equal to zero.

We now define a modified version of $G^A_{\mathcal{P}}$ called $G^A_{\mathcal{P}}(\zeta_1, \zeta_2) = (V^A(\zeta_1, \zeta_2), E^A(\zeta_1, \zeta_2))$ where

$$V^A(\zeta_1, \zeta_2) \stackrel{\text{def}}{=} V^A \cup \{\delta^A_s, \delta^A_z\} \tag{5.1}$$

and

$$E^A(\zeta_1, \zeta_2) \stackrel{\text{def}}{=} E^A \cup \left\{ \begin{array}{l} e^A_s \stackrel{\text{def}}{=} (\delta^A_s, \mathsf{InBlock}(G^A_{\mathcal{P}})), \\ e^A_z \stackrel{\text{def}}{=} (\mathsf{OutBlock}(G^A_{\mathcal{P}}), \delta^A_z) \end{array} \right\} . \tag{5.2}$$

Furthermore, $C(\delta^A_s) = \zeta_1$, $C(\delta^A_z) = \zeta_2$, and $\xi(e^A_s) = \xi(e^A_z) = \infty$. Intuitively, $G^A_{\mathcal{P}}(\zeta_1, \zeta_2)$ is an augmentation of $G^A_{\mathcal{P}}$ by adding non-preemptible BBs δ^A_s and δ^A_z with size ζ_1 and ζ_2 to the beginning and end (respectively) of $G^A_{\mathcal{P}}$. Let $S^A(\zeta_1, \zeta_2) \subseteq E^A(\zeta_1, \zeta_2)$ be the optimal set of EPPs for $G^A_{\mathcal{P}}$ according to the objective function of Equation (2.1) respecting the constraints of Equation (2.2). We will next show that we can obtain an optimal solution to the EPP placement problem for $G_{\mathcal{P}}$ by first optimally solving the EPP placement problem for $G^A_{\mathcal{P}}(\zeta_1, \zeta_2)$ for all productions A in a derivation for $G_{\mathcal{P}}$ and for all values of $\zeta_i \in \{0, 1, \ldots, Q-1\}$ (where $i = 1, 2$).

We first consider production (a):

$$\langle \texttt{Blocks} \rangle \rightarrow (\langle \texttt{SB} \rangle \mid \langle \texttt{CB} \rangle), e_i, \langle \texttt{Blocks} \rangle.$$

In a general application of production (a), we let $G^A_{\mathcal{P}}$ denote the non-terminal block on the left-hand side of the rule, and $G^B_{\mathcal{P}}$ and $G^C_{\mathcal{P}}$ denote the first and second non-terminal blocks, respectively, on the right-hand side of the rule:

$$G^A_{\mathcal{P}} \rightarrow G^B_{\mathcal{P}}, e_i, G^C_{\mathcal{P}},$$

where $e_i = (\mathsf{OutBlock}(G^B_{\mathcal{P}}), \mathsf{InBlock}(G^C_{\mathcal{P}}))$.

Theorem 1. *When applying production (a) over feasible $G_{\mathcal{P}}$ and Q, an optimal set $S^A(\zeta_1, \zeta_2)$ of EPPs for $G^A_{\mathcal{P}}(\zeta_1, \zeta_2)$ (where $\zeta_1, \zeta_2 \in \{0, 1, \ldots, Q-1\}$) is equal to*

$$S^B(\zeta_1, 0) \cup \{e_i\} \cup S^C(\xi(e_i), \zeta_2), \tag{5.3}$$

or one of the following for $x \in \{1, \ldots, Q-1\}$:

$$S^B(\zeta_1, x) \cup S^C(Q - x, \zeta_2). \tag{5.4}$$

Proof. The proof is by contradiction. There are two cases dependent upon whether $e_i \in S^A(\zeta_1, \zeta_2)$ is true. Let us first consider the case where e_i is in the set $S^A(\zeta_1, \zeta_2)$.

Case 1. $[e_i \in S^A(\zeta_1, \zeta_2)]$

Assume there exist $S' \subseteq E^B(\zeta_1, 0)$ and $S'' \subseteq E^C(\xi(e_i), \zeta_2)$, such that $S^A(\zeta_1, \zeta_2)$ equals $S' \cup \{e_i\} \cup S''$, and $\Phi\left(G^A_{\mathcal{P}}(\zeta_1, \zeta_2), S' \cup \{e_i\} \cup S''\right) < \Phi\left(G^A_{\mathcal{P}}(\zeta_1, \zeta_2), S^B(\zeta_1, 0) \cup \{e_i\} \cup S^C(\xi(e_i), \zeta_2)\right)$. In words, $S^B(\zeta_1, 0)$ and $S^C(\xi(e_i), \zeta_2)$ are optimal sets of EPPs for graphs $G^B_{\mathcal{P}}(\zeta_1, 0)$ and $G^C_{\mathcal{P}}(\xi(e_i), \zeta_2)$, respectively, but $S^B(\zeta_1, 0) \cup \{e_i\} \cup S^C(\xi(e_i), \zeta_2)$ is not an optimal set of EPPs for $G^A_{\mathcal{P}}(\zeta_1, \zeta_2)$.

We will show that the following properties are satisfied:

P1: S' and S'' satisfy the constraints of Equation (2.2) for $G^B_{\mathcal{P}}(\zeta_1, 0)$ and $G^C_{\mathcal{P}}(\xi(e_i), \zeta_2)$, respectively.

P2: At least one of the following strict inequalities holds:

$\Phi(G^B_{\mathcal{P}}(\zeta_1, 0), S') < \Phi(G^B_{\mathcal{P}}(\zeta_1, 0), S^B(\zeta_1, 0))$, or

$\Phi(G^C_{\mathcal{P}}(\xi(e_i), \zeta_2), S'') < \Phi(G^C_{\mathcal{P}}(\xi(e_i), \zeta_2), S^C(\xi(e_i), \zeta_2))$.

Property 1 implies that S' and S'' are feasible solutions to the EPP placement problem for $G^B_{\mathcal{P}}(\zeta_1, 0)$ and $G^C_{\mathcal{P}}(\xi(e_i), \zeta_2)$, respectively. Property 2 implies that at least one between $S^B(\zeta_1, 0)$ and $S^C(\xi(e_i), \zeta_2)$ cannot be an optimal solution for $G^B_{\mathcal{P}}(\zeta_1, 0)$ and $G^C_{\mathcal{P}}(\xi(e_i), \zeta_2)$,

respectively, reaching a contradiction.

Proof of P1. To prove Property 1, note that $S' \equiv S^A(\zeta_1, \zeta_2) \cap G^B_{\mathcal{P}}(\zeta_1, 0)$. Since $S^A(\zeta_1, \zeta_2)$ satisfies the EPP constraint for the graph $G^A_{\mathcal{P}}(\zeta_1, \zeta_2)$, and the portion of $G^A_{\mathcal{P}}(\zeta_1, \zeta_2)$ before e_i coincides with $G^B_{\mathcal{P}}(\zeta_1, 0)$ (an exit node of length zero can be simply ignored), then also S' satisfies the EPP constraint of Equation (2.2) for $G^B_{\mathcal{P}}(\zeta_1, 0)$.

Some more steps are needed to prove the same property for S'' and $G^C_{\mathcal{P}}(\xi(e_i), \zeta_2)$. Note that $S'' \equiv S^A(\zeta_1, \zeta_2) \cap G^C_{\mathcal{P}}(\xi(e_i), \zeta_2)$. For each path p entering $G^C_{\mathcal{P}}(\xi(e_i), \zeta_2)$, let $e^p_{\text{first}} = (\delta^p_x, \delta^p_y)$ be the first EPP of S'' in p. Since $S^A(\zeta_1, \zeta_2)$ satisfies the EPP constraint for $G^A_{\mathcal{P}}(\zeta_1, \zeta_2)$, and the portion of $G^A_{\mathcal{P}}(\zeta_1, \zeta_2)$ after δ^p_x coincides with that of $G^C_{\mathcal{P}}(\xi(e_i), \zeta_2)$, all basic blocks of $V^C(\xi(e_i), \zeta_2)$ that come after δ^p_x in path p continue to satisfy the EPP constraint of Equation (2.2) for S'' in $G^C_{\mathcal{P}}(\xi(e_i), \zeta_2)$. Thus, we just need to prove that the remaining basic blocks $(\delta_j \in V^C(\xi(e_i), \zeta_2) : \delta_j \preceq_p \delta^p_x)$ also satisfy the EPP constraint for S'' in $G^C_{\mathcal{P}}(\xi(e_i), \zeta_2)$:

$$\xi((\delta_{-\infty}, \delta^C_s)) + \sum_{\substack{\delta_j \in p \\ \delta^C_s \preceq_p \delta_j \preceq_p \delta^p_x}} C(\delta_j) \leq Q. \tag{5.5}$$

By convention for the sentinel edges, $\xi((\delta_{-\infty}, \delta^C_s))$ equals zero. Since $S'' \cup \{e_i\}$ satisfies the EPP constraint for $G^A_{\mathcal{P}}(\zeta_1, \zeta_2)$, it must be that

$$\xi(e_i) + \sum_{\substack{\delta_j \in p \\ \mathsf{InBlock}(G^C_{\mathcal{P}}) \preceq_p \delta_j \preceq_p \delta^p_x}} C(\delta_j) \leq Q. \tag{5.6}$$

However, observe that $C(\delta^C_s)$ equals $\xi(e_i)$ by definition of $G^C_{\mathcal{P}}(\xi(e_i), \zeta_2)$, and the remaining blocks are $\delta_j \in p : \mathsf{InBlock}(G^C_{\mathcal{P}}) \preceq_p \delta_j \preceq_p \delta_x$. Thus, Equation (5.6) implies that Equation (5.5) is true. Hence, Property 1 holds also for S'' and $G^C_{\mathcal{P}}(\xi(e_i), \zeta_2)$. □

Proof of P2. To prove Property 2, remember that we assumed: $\Phi\left(G^A_{\mathcal{P}}(\zeta_1, \zeta_2), S' \cup \{e_i\} \cup S''\right) < \Phi\left(G^A_{\mathcal{P}}(\zeta_1, \zeta_2), S^B(\zeta_1, 0) \cup \{e_i\} \cup S^C(\xi(e_i), \zeta_2)\right)$.

The considered graph might have multiple paths, all of which pass through e_i. Since

$G^B_{\mathcal{P}}$ and $G^C_{\mathcal{P}}$ are disjoint sets, the smaller Φ function obtained using $S' \cup \{e_i\} \cup S''$ instead of $S^B(\zeta_1, 0) \cup \{e_i\} \cup S^C(\xi(e_i), \zeta_2)$ is due to a better EPP selection in S' than in $S^B(\zeta_1, 0)$, or in S'' than in $S^C(\xi(e_i), \zeta_2)$. In the first case, since the portion of $G^A_{\mathcal{P}}(\zeta_1, \zeta_2)$ before e_i coincides with $G^B_{\mathcal{P}}(\zeta_1, 0)$ (ignoring the exit node of length zero), it follows that $\Phi(G^B_{\mathcal{P}}(\zeta_1, 0), S') < \Phi(G^B_{\mathcal{P}}(\zeta_1, 0), S^B(\zeta_1, 0))$, proving the first inequality of Property 2. In the second case, since the portion of $G^A_{\mathcal{P}}(\zeta_1, \zeta_2)$ after e_i coincides with the portion of $G^C_{\mathcal{P}}(\xi(e_i), \zeta_2)$ after δ^C_s, and $C(\delta^C_s) = \xi(e_i)$, it must be that $\Phi(G^C_{\mathcal{P}}(\xi(e_i), \zeta_2), S'') < \Phi(G^B_{\mathcal{P}}(\xi(e_i), \zeta_2), S^B(\xi(e_i), \zeta_2))$, proving the second inequality of Property 2. □

The case where e_i is in the set $S^A(\zeta_1, \zeta_2)$ stands proved. It remains to prove the other case. □

Case 2. $[e_i \notin S^A(\zeta_1, \zeta_2)]$

The proof is similar to the above case. Assume there exist $S' \subseteq E^B(\zeta_1, x)$ and $S'' \subseteq E^C(Q - x, \zeta_2)$, such that $S^A(\zeta_1, \zeta_2)$ equals $S' \cup S''$, and $\Phi\left(G^A_{\mathcal{P}}(\zeta_1, \zeta_2), S' \cup S''\right) < \Phi\left(G^A_{\mathcal{P}}(\zeta_1, \zeta_2), S^B(\zeta_1, x) \cup S^C(Q - x, \zeta_2)\right)$. In words, $S^B(\zeta_1, x)$ and $S^C(Q - x, \zeta_2)$ are optimal sets of EPPs for graphs $G^B_{\mathcal{P}}(\zeta_1, x)$ and $G^C_{\mathcal{P}}(Q - x, \zeta_2)$, respectively, but $S^B(\zeta_1, x) \cup S^C(Q - x, \zeta_2)$ is not an optimal set of EPPs for $G^A_{\mathcal{P}}(\zeta_1, \zeta_2)$.

It is easy to see that the corresponding versions of properties P1 and P2 of Case 1 also hold in this case:

P1: S' and S'' satisfy the constraints of Equation (2.2) for $G^B_{\mathcal{P}}(\zeta_1, x)$ and $G^C_{\mathcal{P}}(Q - x, \zeta_2)$, respectively.

P2: At least one of the following strict inequalities holds:

$\Phi(G^B_{\mathcal{P}}(\zeta_1, x), S') < \Phi(G^B_{\mathcal{P}}(\zeta_1, x), S^B(\zeta_1, x))$, or

$\Phi(G^C_{\mathcal{P}}(Q - x, \zeta_2), S'') < \Phi(G^C_{\mathcal{P}}(Q - x, \zeta_2), S^C(Q - x, \zeta_2))$.

Since, by P1, S' and S'' are feasible solutions to the EPP placement problem for $G^B_{\mathcal{P}}(\zeta_1, 0)$ and $G^C_{\mathcal{P}}(\xi(e_i), \zeta_2)$, respectively, and, by P2, at least one between $S^B(\zeta_1, 0)$ and

$S^C(\xi(e_i), \zeta_2)$ cannot be an optimal solution for $G^B_{\mathcal{P}}(\zeta_1, 0)$ and $G^C_{\mathcal{P}}(\xi(e_i), \zeta_2)$, respectively, a contradiction is reached, proving also Case 2. □

Having proved that in both considered cases one among Equations (5.3) and (5.4) holds, the Theorem stands proved. □

We have just shown that optimal substructure exists for determining the optimal EPP placement for any subgraph that corresponds to a single block production in a derivation of $G_{\mathcal{P}}$. We can exploit this optimal substructure to obtain a recursive formulation for quantifying the optimal value of Φ for the subgraph. We first define some notation.

Let $\mu_a(\text{expr})$ return a if expr is true and one if false. Moreover, let the function $\mathsf{cost}(A, \zeta_1, \zeta_2)$ be

$$\Phi(G^A_{\mathcal{P}}(\zeta_1, \zeta_2), S^A(\zeta_1, \zeta_2)) - \zeta_1 - \zeta_2,$$

when $G^A_{\mathcal{P}}(\zeta_1, \zeta_2)$ is feasible for Q; otherwise, let $\mathsf{cost}(A, \zeta_1, \zeta_2) = \infty$.

Intuitively, the cost function represents the objective function of the considered subgraph, with artificial weights removed. The artificial weights (ζ_1, ζ_2) are used to model constraints due to non-preemptive regions overlapping with the start or the end of the considered subgraph, i.e., they represent the non-preemptive carry-in and carry-out of the considered subgraph. When computing the WCET+CRPD of a subgraph in our dynamic programming formulation, such artificial weights will therefore be subtracted.

In the following, we show how to compute the cost function for all valid productions in our grammar $\mathcal{G}$. For each rule, we will provide a cost matrix spanning all meaningful artificial weights: $\zeta_1, \zeta_2 \in \{0, 1, \ldots, Q-1\}$.

For a production (a), the cost matrix can be computing using the following corollary to Theorem 1.

Corollary 1. *When applying production (a) over feasible $G_{\mathcal{P}}$ and Q, the* cost *matrix for the optimal $S^A(\zeta_1, \zeta_2)$ of EPPs for $G^A_{\mathcal{P}}(\zeta_1, \zeta_2)$ (where $\zeta_1, \zeta_2 \in \{0, 1, \ldots, Q-1\}$) can be*

constructed by the following recursive computation:

$$\mathsf{cost}\left(\left[G^A_{\mathcal{P}} \to G^B_{\mathcal{P}}, e_i, G^C_{\mathcal{P}}\right], \zeta_1, \zeta_2\right) \stackrel{def}{=} \min \left\{ \begin{array}{l} \mathsf{cost}(G^B_{\mathcal{P}}, \zeta_1, 0) + \xi(e_i) + \mathsf{cost}(G^C_{\mathcal{P}}, \xi(e_i), \zeta_2), \\ \min\limits_{x=1}^{Q-1} \left\{ \mathsf{cost}(G^B_{\mathcal{P}}, \zeta_1, x) + \mathsf{cost}(G^C_{\mathcal{P}}, Q-x, \zeta_2) \right\} \end{array} \right\}. \tag{5.7}$$

For a production (b) deriving either a conditional or a single block ($\langle \texttt{Blocks} \rangle \to \langle \texttt{SB} \rangle | \langle \texttt{CB} \rangle$), the cost function is

$$\mathsf{cost}\left(\left[G^A_{\mathcal{P}} \to \left(G^{seq}_{\mathcal{P}} \mid G^{con}_{\mathcal{P}}\right)\right], \zeta_1, \zeta_2\right) \stackrel{\text{def}}{=} \mathsf{cost}((G^{seq}_{\mathcal{P}} \mid G^{con}_{\mathcal{P}}), \zeta_1, \zeta_2). \tag{5.8}$$

For a production (c), instantiating a single basic block ($\langle \texttt{SB} \rangle \to \delta_i$), the cost function is

$$\mathsf{cost}\left([\langle \texttt{SB} \rangle \to \delta_i], \zeta_1, \zeta_2\right) \stackrel{\text{def}}{=} \mu_\infty \left(\zeta_1 + \zeta_2 + C(\delta_i) > Q\right) \cdot C(\delta_i). \tag{5.9}$$

The following theorem and corollary pertain to the application of production (d), for a conditional block structure $\langle \texttt{CB} \rangle$:

$$\langle \texttt{CB} \rangle \to \delta_i, [e_{i_1}, \langle \texttt{Blocks} \rangle, e_{j_1}], ([e_{i_2}, \langle \texttt{Blocks} \rangle, e_{j_2}]) +, \delta_j.$$

In a general application of production (d), we let $G^A_{\mathcal{P}}$ denote the conditional block on the left-hand side of the rule, and $G^{B_1}_{\mathcal{P}}, \ldots, G^{B_k}_{\mathcal{P}}$ denote the parallel non-terminal blocks on the right-hand side of the rule:

$$G^A_{\mathcal{P}} \to \delta_i, [e_{i_1}, G^{B_1}_{\mathcal{P}}, e_{j_1}], \ldots, [e_{i_k}, G^{B_k}_{\mathcal{P}}, e_{j_k}], \delta_j.$$

where $e_{i_\ell} = (\delta_i, \mathsf{InBlock}(B_\ell))$, and $e_{j_\ell} = (\mathsf{OutBlock}(B_\ell), \delta_j)$ for $\ell = 1, \ldots, k$. Moreover, we let $G^{T_\ell}_{\mathcal{P}}$ denote the subgraph composed of $G^{B_\ell}_{\mathcal{P}}$, and basic blocks δ_i and δ_j as the first and last block, respectively.

Theorem 2. *When applying production (d) over feasible $G_{\mathcal{P}}$ and Q, an optimal set $S^A(\zeta_1, \zeta_2)$ of EPPs for $G^A_{\mathcal{P}}(\zeta_1, \zeta_2)$ (where $\zeta_1, \zeta_2 \in \{0, 1, \ldots, Q-1\}$) is equal to*

$$\bigcup_{\ell=1}^{k} S^{T_\ell}(\zeta_1, \zeta_2), \tag{5.10}$$

where for each $\ell \in \{1, \ldots, k\}$, the optimal EPP set $S^{T_\ell}(\zeta_1, \zeta_2)$ equals one of the following sets:

$$S^{B_\ell}(\zeta_1 + C(\delta_i), \zeta_2 + C(\delta_j)), \tag{5.11}$$

$$S^{B_\ell}(\xi(e_{i_\ell}), \zeta_2 + C(\delta_j)) \cup \{e_{i_\ell}\}, \tag{5.12}$$

$$S^{B_\ell}(\zeta_1 + C(\delta_i), 0) \cup \{e_{j_\ell}\}, \tag{5.13}$$

$$S^{B_\ell}(\xi(e_{i_\ell}), 0) \cup \{e_{i_\ell}, e_{j_\ell}\}. \tag{5.14}$$

Proof. The same technique adopted in Case 1 and Case 2 of Theorem 1 is here applied to each parallel subgraph $G^{T_\ell}_{\mathcal{P}}(\zeta_1, \zeta_2)$, for $\ell \in \{1, \ldots, k\}$. For each such subgraph, four different cases are considered:

- $e^\ell_i \notin S^A(\zeta_1, \zeta_2)$, $e^\ell_j \notin S^A(\zeta_1, \zeta_2)$, leading to Eq. (5.11);
- $e^\ell_i \in S^A(\zeta_1, \zeta_2)$, $e^\ell_j \notin S^A(\zeta_1, \zeta_2)$, leading to Eq. (5.12);
- $e^\ell_i \notin S^A(\zeta_1, \zeta_2)$, $e^\ell_j \in S^A(\zeta_1, \zeta_2)$, leading to Eq. (5.13);
- $e^\ell_i \in S^A(\zeta_1, \zeta_2)$, $e^\ell_j \in S^A(\zeta_1, \zeta_2)$, leading to Eq. (5.14).

The optimal set $S^A(\zeta_1, \zeta_2)$ is simply the union of all subgraphs solutions.

In the following, we will prove the first and fourth cases since the proofs of the second and third cases are very similar. We prove the theorem by contradiction. That is, assume that we can obtain a better for $G^A_{\mathcal{P}}(\zeta_1, \zeta_2)$ by not using the optimal sub-solution for some branch ℓ. We now consider (case-by-case) whether e^ℓ_i an e^ℓ_j are part of the optimal solution for $G^A_{\mathcal{P}}(\zeta_1, \zeta_2)$ and show that this leads to a contradiction in each case.

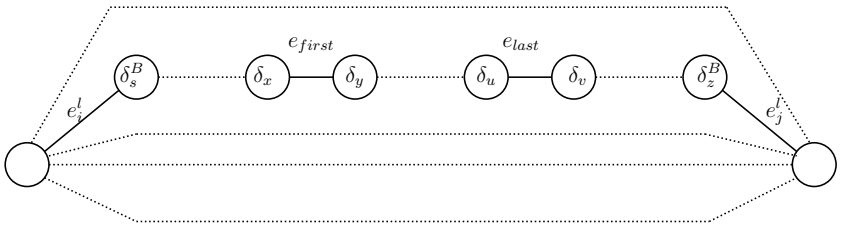


Figure 5.2: Notations of each path p in production(d).

Case 1. $[e_i^\ell \notin S^A(\zeta_1, \zeta_2), e_j^\ell \notin S^A(\zeta_1, \zeta_2)]$

Assume there exist $S' \subseteq E^{B_\ell}(\zeta_1 + C(\delta_i), \zeta_2 + C(\delta_j))$, such that $S^A(\zeta_1, \zeta_2)$ equals S', and $\Phi\left(G^A_{\mathcal{P}}(\zeta_1, \zeta_2), S'\right) < \Phi\left(G^A_{\mathcal{P}}(\zeta_1, \zeta_2), S^{B_\ell}(\zeta_1 + C(\delta_i), \zeta_2 + C(\delta_j))\right)$. In words, $S^{B_\ell}(\zeta_1 + C(\delta_i), \zeta_2 + C(\delta_j))$ is optimal set of EPPs for graphs $G^{B_\ell}_{\mathcal{P}}(\zeta_1 + C(\delta_i), \zeta 2 + C(\delta_j))$, but $S^{B_\ell}(\zeta_1 + C(\delta_i), \zeta_2 + C(\delta_j))$ is not an optimal set of EPPs for $G^A_{\mathcal{P}}(\zeta_1, \zeta_2)$.

We will show that the following properties are satisfied:

P1: S' satisfies the constraints of Equation (2.2) for $G^{B_\ell}_{\mathcal{P}}(\zeta_1 + C(\delta_i), \zeta_2 + C(\delta_j))$.

P2: $\Phi(G^{B_\ell}_{\mathcal{P}}(\zeta_1, \zeta_2), S') < \Phi(G^{B_\ell}_{\mathcal{P}}(\zeta_1 + C(\delta_i), \zeta_2 + C(\delta_j)), S^{B_\ell}(\zeta_1 + C(\delta_i), \zeta_2 + C(\delta_j)))$.

Property 1 implies that S' is a feasible solution to the EPP placement problem for $G^{B_\ell}_{\mathcal{P}}(\zeta_1 + C(\delta_i), \zeta_2 + C(\delta_j))$. Property 2 implies that $S^{B_\ell}(\zeta_1 + C(\delta_i), \zeta_2 + C(\delta_j))$ cannot be an optimal solution for $G^{B_\ell}_{\mathcal{P}}(\zeta_1 + C(\delta_i), \zeta_2 + C(\delta_j))$, reaching a contradiction.

Proof of P1. Regarding Property 1, let the set of basic blocks $V_{\text{left}} \subseteq V^{B_\ell}(\zeta_1 + C(\delta_i), \zeta_2 + C(\delta_j))$ that have the closest preceding EPP in S' equal to the sentinel edge $(\delta_{-\infty}, \delta_s^B)$, and the set of basic blocks $V_{\text{right}} \subseteq V^{B_\ell}(\zeta_1 + C(\delta_i), \zeta_2 + C(\delta_j))$ that have the closest succeeding EPP in S' equal to the sentinel edge $(\delta_z^B, \delta_\infty)$. Clearly, all basic blocks in $V^{B_\ell}(\zeta_1 + C(\delta_i), \zeta_2 + C(\delta_j)) \setminus (V_{\text{left}} \cup V_{\text{right}}))$ continue to satisfy the EPP constraint of Equation 2.2 for S' in $G^A_{\mathcal{P}}(\zeta_1, \zeta_2)$ since $S^A(\zeta_1, \zeta_2)$ satisfies the EPP constraint (Equation 2.2) for $G^A_{\mathcal{P}}(\zeta_1, \zeta_2)$ and this portion is identical in the two graphs. Thus, we just need to prove that the $\delta_\ell \in (V_{\text{left}} \cup V_{\text{right}})$ basic blocks still satisfy the EPP constraint for S' in $G^{B_\ell}_{\mathcal{P}}(\zeta_1 + C(\delta_i), \zeta_2 + C(\delta_j))$. Let $e_{\text{first}} = (\delta_x, \delta_y)$ be the first EPP of S' and $e_{\text{last}} = (\delta_u, \delta_v)$ be the last EPP

of S'. Thus, we may prove the EPP constraint if we can show that the following two inequalities are true.

$$\xi((\delta_{-\infty}, \delta_s^B)) + \sum_{\substack{\delta_k \in \ell \\ \delta_s^B \preceq_\ell \delta_k \preceq_\ell \delta_x}} C(\delta_k) \leq Q. \tag{5.15}$$

$$\xi((\delta_u, \delta_v)) + \sum_{\substack{\delta_k \in \ell \\ \delta_v \preceq_\ell \delta_k \preceq_\ell \delta_z^B}} C(\delta_k) + C(\delta_\infty) \leq Q. \tag{5.16}$$

Since S' satisfies the EPP constraint for $G_{\mathcal{P}}^A(\zeta_1, \zeta_2)$, it must be that

$$\xi((\delta_{-\infty}, \delta_s^A)) + \sum_{\substack{\delta_k \in \ell \\ \delta_s^A \preceq_\ell \delta_k \preceq_\ell \delta_x}} C(\delta_k) \leq Q. \tag{5.17}$$

$$\xi((\delta_u, \delta_v)) + \sum_{\substack{\delta_k \in \ell \\ \delta_v \preceq_\ell \delta_k \preceq_\ell \delta_z^A}} C(\delta_k) + C(\delta_\infty) \leq Q. \tag{5.18}$$

Observe that both $\xi((\delta_{-\infty}, \delta_s^A))$ and $\xi((\delta_{-\infty}, \delta_s^B))$ equal zero, $C(\delta_s^B)$ equals $C(\delta_i) + C(\zeta_1)$ which is $C(\delta_s^A) + C(\zeta_1)$, and $C(\delta_z^B)$ equals $C(\delta_j) + C(\zeta_2)$ which is $C(\delta_z^A) + C(\zeta_2)$. Thus, Equation 5.15 and Equation 5.16 are true. Hence, Property 1 holds for $G_{\mathcal{P}}^{B_\ell}(\zeta_1 + C(\delta_i), \zeta_2 + C(\delta_j))$ and S'.

□

Regarding Property 2, we assumed that $\Phi(G_{\mathcal{P}}^A(\zeta_1, \zeta_2), S') < \Phi(G_{\mathcal{P}}^A(\zeta_1, \zeta_2), S^{B_\ell}(\zeta_1 + C(\delta_i), \zeta_2 + C(\delta_j)))$. Thus, observe that $\Phi(G_{\mathcal{P}}^A(\zeta_1, \zeta_2), S^{B_\ell}(\zeta_1 + C(\delta_i), \zeta_2 + C(\delta_j))$ equals $\Phi(G_{\mathcal{P}}^{B_\ell}(\zeta_1 + C(\delta_i), \zeta_2 + C(\delta_j)), S^{B_\ell}(\zeta_1 + C(\delta_i), \zeta_2 + C(\delta_j)))$. Therefore, Property 2 holds and the Theorem is true for the first case of $e_i \notin S^A(\zeta_1, \zeta_2), e_j \notin S^A(\zeta_1, \zeta_2)$.

□

Case 4. $[e_i^\ell \in S^A(\zeta_1, \zeta_2), e_j^\ell \in S^A(\zeta_1, \zeta_2)]$

Assume there exists an $S' \subseteq E^{B_\ell}(\xi(e_{i_\ell}), \xi(e_{j_\ell}))$ such that $S^A(\zeta_1, \zeta_2)$ equals $S' \cup \{e_i^\ell, e_j^\ell\}$

and $\Phi\left(G^A_{\mathcal{P}}(\zeta_1,\zeta_2), S' \cup \{e^\ell_i, e^\ell_j\}\right) < \Phi\left(G^A_{\mathcal{P}}(\zeta_1,\zeta_2), S^{B_\ell}(\xi(e_{i_\ell}),\xi(e_{j_\ell})) \cup \{e^\ell_i, e^\ell_j\}\right)$. In words, $S^{B_\ell}(\xi(e_{i_\ell}),\xi(e_{j_\ell}))$ is optimal set of EPPs for graph $G^{B_\ell}_{\mathcal{P}}(\xi(e_{i_\ell}),\xi(e_{j_\ell}))$, but $S^{B_\ell}(\xi(e_{i_\ell}),\xi(e_{j_\ell})) \cup \{e^\ell_i, e^\ell_j\}$ is not an optimal set of EPPs for $G^A_{\mathcal{P}}(\zeta_1,\zeta_2)$. We will show that S' satisfies the following three properties:

P 1: S' satisfies the constraints of Equation 2.2 for $G^{B_\ell}_{\mathcal{P}}(\xi(e_{i_\ell}),\xi(e_{j_\ell}))$.

P 2: $\Phi(G^{B_\ell}_{\mathcal{P}}(\xi(e_{i_\ell}),\xi(e_{j_\ell})), S')$ is strictly less than $\Phi(G^{B_\ell}_{\mathcal{P}}(\xi(e_{i_\ell}),\xi(e_{j_\ell})), S^{B_\ell}(\xi(e_{i_\ell}),\xi(e_{j_\ell})))$.

Property 1 implies that S' is a feasible solution to the EPP placement problem for $G^{B_\ell}_{\mathcal{P}}(\xi(e_{i_\ell}),\xi(e_{j_\ell}))$. Property 2 implies that $S^{B_\ell}(\xi(e_{i_\ell}),\xi(e_{j_\ell}))$ is not the optimal solution for $G^{B_\ell}_{\mathcal{P}}(\xi(e_{i_\ell}),\xi(e_{j_\ell}))$; this is a contradiction to the definition of $S^{B_\ell}(\xi(e_{i_\ell}),\xi(e_{j_\ell}))$. In this case, our assumption of the existence of S' is false and $S^A(\zeta_1,\zeta_2)$ equals $S^{B_\ell}(\xi(e_{i_\ell}),\xi(e_{j_\ell})) \cup \{e^\ell_i, e^\ell_j\}$, thus proving the theorem for the first case. In the following, we prove Properties 1-2.

Proof of P1. Regarding Property 1, observe that $G^{B_\ell}_{\mathcal{P}}(\xi(e_{i_\ell}),\xi(e_{j_\ell}))$ differs from $G^A_{\mathcal{P}}(\zeta_1,\zeta_2)$ in the addition of δ_i and δ_j, and the differing cost of the δ_s and δ_z. Furthermore, S' and $S^A(\zeta_1,\zeta_2)$ differ in the edge e^ℓ_i and e^ℓ_j by definition of S'. Thus, since $S^A(\zeta_1,\zeta_2)$ is a solution to the EPP problem, it must include sentinel edges $(\delta_{-\infty},\delta_s)$ and (δ_z,δ_∞). S' must also include these sentinel edges. Let the set of basic blocks $V_{\text{left}} \subseteq V^{B_\ell}(\xi(e_{i_\ell}),\xi(e_{j_\ell}))$ that have the closest preceding EPP in S' equal to the sentinel edge $(\delta_{-\infty},\delta^B_s)$, and the set of basic blocks $V_{\text{right}} \subseteq V^{B_\ell}(\xi(e_{i_\ell}),\xi(e_{j_\ell}))$ that have the closest succeeding EPP in S' equal to the sentinel edge $(\delta^B_z,\delta_\infty)$. Note that the set of basic blocks $V_{\text{right}} \subseteq V^{B_\ell}(\xi(e_{i_\ell}),\xi(e_{j_\ell}))$ have the closest succeeding EPP in $S^A(\zeta_1,\zeta_2)$ equal to the edge e_j. Clearly, all basic blocks in $V^{B_\ell}(\xi(e_{i_\ell}),\xi(e_{j_\ell})) \backslash (V_{\text{left}} \cup V_{\text{right}})$ continue to satisfy the EPP constraint of Equation 2.2 for S' in $G^A_{\mathcal{P}}(\zeta_1,\zeta_2)$ since $S^A(\zeta_1,\zeta_2)$ satisfies the EPP constraint (Equation 2.2) for $G^A_{\mathcal{P}}(\zeta_1,\zeta_2)$ and this portion is identical in the two graphs. Thus, we just need to prove that the $\delta_\ell \in (V_{\text{left}} \cup V_{\text{right}})$ basic blocks still satisfy the EPP constraint for S' in $G^{B_\ell}_{\mathcal{P}}(\xi(e_{i_\ell}),\xi(e_{j_\ell}))$. Let δ_{e_j} be the BB right before e^ℓ_j, $e_{\text{first}} = (\delta_x,\delta_y)$ be the first EPP of S' and $e_{\text{last}} = (\delta_u,\delta_v)$

be the last EPP of S'. Thus, we may prove the EPP constraint if we can show that the following two inequalities are true.

$$\xi((\delta_{-\infty}, \delta_s^B)) + \sum_{\substack{\delta_k \in \ell \\ \delta_s^B \preceq_\ell \delta_k \preceq_\ell \delta_x}} C(\delta_k) \leq Q. \tag{5.19}$$

$$\xi((\delta_u, \delta_v)) + \sum_{\substack{\delta_k \in \ell \\ \delta_v \preceq_\ell \delta_k \preceq_\ell \mathsf{OutBlock}(G_{\mathcal{P}}^{B_\ell})}} C(\delta_k) \leq Q. \tag{5.20}$$

Since $S' \cup \{e_i^\ell, e_j^\ell\}$ satisfies the EPP constraint for $G_{\mathcal{P}}^A(\zeta_1, \zeta_2)$, it must be that

$$\xi(e_{i_\ell}) + \sum_{\substack{\delta_k \in \ell \\ \mathsf{InBlock}(B) \preceq_\ell \delta_k \preceq_\ell \delta_x}} C(\delta_k) \leq Q. \tag{5.21}$$

$$\xi((\delta_u, \delta_v)) + \sum_{\substack{\delta_k \in \ell \\ \delta_v \preceq_\ell \delta_k \preceq_\ell \delta_{e_j}}} C(\delta_k) \leq Q. \tag{5.22}$$

Observe that $C(\delta_s^B)$ equals $\xi(e_{i_\ell})$, and $C(\delta_z^B)$ equals $\xi(e_{j_\ell})$ by definition of $G_{\mathcal{P}}^{B_\ell}(\xi(e_{i_\ell}), \xi(e_{j_\ell}))$. δ_{e_j} is the same as $\mathsf{OutBlock}(G_{\mathcal{P}}^{B_\ell})$ in $G_{\mathcal{P}}^{B_\ell}(\xi(e_{i_\ell}), \xi(e_{j_\ell}))$. Thus, Equation 5.19 and Equation 5.20 are true. Hence, Property 1 holds for $G_{\mathcal{P}}^{B_\ell}(\xi(e_{i_\ell}), \xi(e_{j_\ell}))$ and S'.

□

Regarding Property 2, in the first paragraph of the proof, we assumed that $\Phi(G_{\mathcal{P}}^A(\zeta_1, \zeta_2), S' \cup \{e_i^\ell, e_j^\ell\}) < \Phi(G_{\mathcal{P}}^A(\zeta_1, \zeta_2), S^{B_\ell}(\xi(e_{i_\ell}), \xi(e_{j_\ell})) \cup \{e_i^\ell, e_j^\ell\})$. Observe that for any $S \subseteq E^{B_\ell}(\xi(e_{i_\ell}), \xi(e_{j_\ell}))$ where $e_i \notin S$,$e_j \notin S$, it is easy to show that $\Phi(G_{\mathcal{P}}^A(\zeta_1, \zeta_2), S' \cup \{e_i^\ell, e_j^\ell\})$ equals $\Phi(G_{\mathcal{P}}^{B_\ell}(\xi(e_{i_\ell}), \xi(e_{j_\ell})), S')$, since the removal of e_i^ℓ and e_j^ℓ is offset by setting $C(\delta_s^B)$ equal to $\xi(e_{i_\ell})$ and $C(\delta_z^B)$ equal to $\xi(e_{j_\ell})$, respectively. For identical reasons, $\Phi(G_{\mathcal{P}}^A(\zeta_1, \zeta_2), S^{B_\ell}(\xi(e_{i_\ell}), \xi(e_{j_\ell})) \cup \{e_i^\ell, e_j^\ell\})$ equals $\Phi(G_{\mathcal{P}}^{B_\ell}(\xi(e_{i_\ell}), \xi(e_{j_\ell})), S^{B_\ell}(\xi(e_{i_\ell}), \xi(e_{j_\ell})))$. Therefore, Property 2 holds and the Theorem is true for the third case of $e_i^\ell \in S^A(\zeta_1, \zeta_2), e_j^\ell \in S^A(\zeta_1, \zeta_2)$.

□

Thus, the Theorem stands proved.

□

Using the above Theorem, the following method is derived to compute the cost function for production (d).

Corollary 2. *When applying production (d) over feasible $G_{\mathcal{P}}$ and Q, the* cost *matrix for the optimal $S^A(\zeta_1, \zeta_2)$ of EPPs for $G^A_{\mathcal{P}}(\zeta_1, \zeta_2)$ (where $\zeta_1, \zeta_2 \in \{0, 1, \ldots, m-1\}$) can be constructed by the following recursive computation:*

$$\mathsf{cost}\left(\left[G^A_{\mathcal{P}} \to \delta_i, [e_{i_1}, G^{B_1}_{\mathcal{P}}, e_{j_1}], \ldots, [e_{i_k}, G^{B_k}_{\mathcal{P}}, e_{j_k}], \delta_j\right], \zeta_1, \zeta_2\right)$$

$$\stackrel{def}{=} \max_{\ell \in \{1,\ldots,k\}} \left\{ \min_{\substack{\alpha \in \{0,1\} \\ \beta \in \{0,1\}}} \left\{ \begin{array}{l} \mathsf{cost}\left(\langle \mathtt{SB} \rangle \equiv [\delta_i], \zeta_1, 0\right) \\ \quad +\mu_0(\alpha = 0) \cdot \xi(e_{i_\ell}) \\ \quad +\mathsf{cost}\big(G^{B_\ell}_{\mathcal{P}}, \\ \qquad \mu_0(\alpha = 1) \cdot (\zeta_1 + C(\delta_i)), \\ \qquad \mu_0(\beta = 1)(\zeta_2 + C(\delta_j))\big) \\ \quad +\mu_0(\beta = 0) \cdot \xi(e_{j_\ell}) \end{array} \right\} \right\}. \tag{5.23}$$

The following theorem and corollary pertain to the optimal substructure and computation for a ⟨`Blocks`⟩ production which combines either a conditional or sequential block with another ⟨`Blocks`⟩ production.

5.2 Algorithm Overview

A standard dynamic-programming algorithm follows directly from the formal statements of Corollaries 1 and 2. However, in this subchapter, we will give a high-level intuitive overview of the algorithm. The algorithm has two major phases:

Parsing: In this phase, a lexical analyzer will break the input graph into a stream of "tokens" (i.e., the basic components of the input graphs). The stream of tokens is used as input into the semantic analyzer which applies the production rules of Figure 4.1 to

produce a derivation. The derivation for an input graph can be represented by a *parse tree* where each internal node of the tree represents a non-terminal symbol and each leaf is a terminal symbol (e.g., a basic block or edge between a basic block). In the resulting parse tree, a node y is the child of node x when y appears in the left-hand-side of some derivation of x. For instance, Figure 4.2 can be viewed as a parse tree where the parent-child relation in the parse tree is equivalent to a box being immediately contained inside another box. As an example, the ⟨SB⟩ node in Figure 4.2 contains four children: basic blocks C and F (which are also leaf nodes in the parse tree) and two ⟨Blocks⟩ nodes (which have their own children).

Bottom-up cost Matrix Computation: Using the parse-tree as input, our algorithm determines the $Q \times Q$ values of the cost matrix for every node in the parse tree by applying the appropriate computational rules given in the previous subchapter. Pseudocode for determining the cost matrix for each of the nodes is provided in Chapter5.3.1. Since the parse tree is traversed in a bottom-up fashion, each node (except the leaves) can reuse the cost matrix of its children nodes when it is determining its own cost matrix.

After applying the approach above, we can obtain the minimum worst-case execution time for the input graph over all possible sets of EPPs that satisfy Equation (2.2) by looking at the cost[0][0] entry for the root node of the parse tree.

5.3 Implementation

For this project, we have utilized a grammar development environment called ANTLRWorks (Version 3) [1]. ANTLRWorks is a compiler generator (similar to Lex/Flex or Yacc/Bison) and can automatically generate a compiler from a grammar file. For the implementation of our algorithm (which can be viewed as a compiler for control flowgraphs), we specified the production rules of Figure 4.1 in a context-free grammar. As mentioned in Chapter 4, our grammar is slightly different than the one presented in the thesis to make it unambiguous. Furthermore, we modified the grammar to ensure that it is an

LL(k) grammar which means that it parses from left-to-right and constructs the leftmost derivation first using only a finite number of "look-ahead" tokens to determine which is the next production rule to apply. (See a textbook on languages such as Scott [22] for more details on grammars and compilers). We develop an LL(k) as it is known to be linear in the number of tokens in an input stream. Furthermore, in ANTLRWorks, we can embed, into the grammar file, the dynamic programming algorithms which we have written in Java. As the parser creates the parse-tree using recursive descent, it also automatically applies the appropriate dynamic-programming functions along the way to compute the cost matrices as it returns up the parse tree.

5.3.1 Graph Grammar

Figure 5.3 illustrates the pseudocode of the context-free grammar. The bold words are production rules, and the words in the curly braces are corresponding embedded actions. The production rules extended from Figure 4.1 are written in EBNF syntax notations. The grammar starts with **prog** which is represented by **q** and **blocks**. The **NEWLINE** works as a delimiter which does not have a special meaning in the graph grammar. In Line 3, **blocks** could either be led by a sequential block or a conditional block. **sb_lead_blocks** comprises a sequential block and is followed by **blocks**. The non-terminal **blocks** here make the grammar have a recursive structure in conditional blocks. Similarly, **cb_lead_blocks** comprises a conditional block and **blocks**. "?" and "+" outside paired parentheses function like loops. The difference between them is that "?" allows the rules in parentheses to execute 0 time, but "+" requires that the rules in parentheses to execute at least one time. In the derivation of **cb**, the first two **bb**s indicate the forking basic block and joining basic block. The two **INT** indicate the edge value before and after the **blocks**. The contents in quotes are the formats which the input must follow. The structure of the grammar follows the top down style. Starting from the **prog**, the grammar checks whether the input has **blocks**, then goes to **blocks**

GRAPH GRAMMAR

```
1  prog : q blocks NEWLINE;
2  q : INT;
3  blocks : sb_lead_blocks | cb_lead_blocks;
4  sb_lead_blocks : sb (INT cb (INT blocks)?)? {EPP_Blocks_Select(lb, rb, e, Q);};
5  cb_lead_blocks : cb (INT blocks)? {EPP_Blocks_Select(lb, rb, e, Q);};
6  cb : '[' bb bb ('<' INT blocks INT '>') + ']' {EPP_CB_Select(Q, δ_i, δ_j, cb);};
7  sb : bb (INT sb)? {EPP_SB_Select(δ_i, Q);};
8  bb : '(' ID ',' INT ')';
```

Figure 5.3: Psuedocode for the graph grammar.

EPP_SB_SELECT(BasicBlock bb, int Q)

```
   ▷ Dynamic-Programming Code for Production in Figure 4.1(c)
1  for i ← 1 to Q
2        do for j ← 1 to Q
3              do if (i + j + C(bb) < Q)
4                    then bb.cost[i][j] = C(bb)
5                    else
                           ▷ Infeasible: set cost entry to some large value
6                          bb.cost[i][j] = MAX
```

Figure 5.4: Psuedocode for the Dynamic-Programming Algorithms for Sequential Block.

to see whether the input is **sb_lead_blocks** or not. Straight down to the **bb**, the parser stores the information of **bb** and then returns the matrix upwards step by step. At last, the code returns the cost of the WCET+CRPD of the whole graph.

5.3.2 Corresponding Embedded Action Function

Figures 5.4, 5.5, and 5.6 are the Java functions that are called by the embedded actions in graph grammar. In *EPP_SB_Select* function, the matrix goes through all possible $Q \times Q$ possibilities to store the costs for each basic block. If the value of $(i + j + C(bb))$ violates the condition in Line 3, then the matrix will store some MAX constant used to represent an infeasible solution in $cost[i][j]$.

```
EPP_CB_SELECT(Q, δ_i, δ_j, CBcb)
    ▷ Dynamic-Programming Code for Production in Figure 4.1(d)
 1  Initialize : cb.cost[][] = 0
 2  for x ← 1 to Q
 3        do for y ← 1 to Q
 4           maxcost ← 0
 5           for k ← 1 to Number of Branches in cb
             ▷ For each branch in the conditional block
 6                 do Set BLK to be the block corresponding to the k'th branch
 7                       do temp1 = C(δ_i) + BLK.cost[x + C(δ_i)][y + C(δ_j)]
 8                          temp2 = C(δ_i) + ξ(e_{i,1}) + BLK.cost[ξ(e_{i,1})][y + C(δ_j)]
 9                          temp3 = C(δ_i) + BLK.cost[x + C(δ_i)][0]
10                          temp4 = C(δ_i) + ξ(e_{i,1}) + BLK.cost[ξ(e_{i,1})][0]
11                          tempcost = min(temp1, temp2, temp3, temp4)
12                    if tempcost > maxcost
13                       then maxcost ← tempcost
14           cb.cost[x][y] ← maxcost
```

Figure 5.5: Psuedocode for the Dynamic-Programming Algorithms for Conditional Block.

A high-level description of the conditional block follows for all possible $Q \times Q$ costs. First, find all the minimum overhead of four possible situations in each sequential branch using the *EPP_SB_Select* function, then choose the max of minimums as the final cost of *cb.cost*[*i*][*j*]. In Step 1, from Lines 7 to 10, there are four possible ways depending on whether we take preemptions at e_i^l or e_j^l for each path in Figure 5.2. The variable x (the first **INT**) is the WCET of e_i^l , and y (the second **INT**) is the WCET of e_j^l. We select the min value of this four in Line 11 as the temporary value of *cb.cost*[*i*][*j*], then choose the max temporary value of all branches as the final value of *cb.cost*[*i*][*j*].

In Figure 5.6, the function combines sub-blocks into a bigger block. The sub-blocks can be either a sequential block or a conditional block. In this code, the inputs are a left block *lb* and a right block *rb*. The main idea is to combine the two blocks in to a bigger one. In the comparison between CostWEdge and CostWOEdge, we choose the smaller one as the cost of *cost*[*i*][*j*]. CostWEdge is the cost with edge, which means that we take

```
EPP_Blocks_Select(Block lb, Block rb, edge e, int Q)
    ▷ Dynamic-Programming Code for Production in Figure 4.1(a)
    ▷ e is the edge connecting the two blocks lb and rb
 1  for i ← 1 to Q
 2       do for j ← 1 to Q
 3             do CostWEdge = lb.cost[i][0] + rb.cost[ξ(e)][j]
 4                CostWOEdge = MAX
 5                for k ← 1 to Q
 6                      do temp = lb.cost[i][k] + rb.cost[Q − k][j]
 7                         if (temp < CostWOEdge)
 8                            then CostWOEdge = temp
 9                if (CostWEdge < CostWOEdge)
10                   then this.cost[i][j] = CostWEdge
11                   else  this.cost[i][j] = CostWOEdge
```

Figure 5.6: Psuedocode for the Dynamic-Programming Algorithms for Block.

the preemption at the edge between the left block and right block. From Lines 5 to 8, in order to get the value of CostWOEdge, we calculate the smallest sum of $lb.cost[i][k]$ and $rb.cost[Q-k][j]$ by traversing k from 1 to Q. The correctness of CostWOEdge is already proved in Theorem 1 and Theorem 2.

5.4 Computational Complexity

The computational complexity of our algorithm is determined by the complexity of generating the parse tree and the complexity of computing the cost matrix for each node. Per the discussion above, the complexity of a parse tree is linear in the number of tokens (which is proportional to $|V|$ for the input graph $G_{\mathcal{P}}$). For the complexity of generating the cost matrix for each node of the parse tree, we need to consider the complexity of applying each rule. Given that constructing the rule in Figure 4.1(b) (i.e., $\langle \texttt{Blocks} \rangle \rightarrow (\langle \texttt{SB} \rangle | \langle \texttt{CB} \rangle), e_i, \langle \texttt{Blocks} \rangle$) is the most computationally expensive and there are at most $|V|$ nodes in the parse tree, the total time complexity of the implementation described above is $O(|V|Q^3)$. Thus, the runtime of our algorithm is pseudo-polynomial

time since it depends upon the value of Q.

§An Alternative Heuristic. The above time complexity is potentially still quite large if Q is a large number. Specifically, in practice, the window of non-preemption (i.e., Q) could be potentially much larger than the basic block size. Therefore, it is worthwhile to find an algorithm that does not depend upon the value of Q. In the algorithm described above, for each node A of the parse tree, we compute $Q \times Q$ different solutions for the node. The reason for storing Q^2 values is that at the time we are solving the subproblem corresponding to node A (i.e., $G_{\mathcal{P}}^A$), it has not been determined when the nearest preemption points before or after $G_{\mathcal{P}}^A$ will occur. Thus, we compute all possible combinations. However, it can be shown that using a larger value of preemption points before or after $G_{\mathcal{P}}^A$ will only increase the length of the longest path in $G_{\mathcal{P}}^A$ since more EPPs will need to be selected in the subgraph in order to satisfy the constraint of Equation 2.2. Thus, as a way to reduce the complexity of the exact approach by potentially obtaining a slight overestimate in the minimum EPP selection, we may fix the dimensions of the cost matrix to be a constant $\alpha \times \alpha \ : \ \alpha \in \mathbb{N}^+$. With this approach, we are only storing values of preemptions before and after each $G_{\mathcal{P}}^A$ equal to $i \cdot \lceil \frac{Q}{\alpha} \rceil$ where $i = 0, 1, \ldots, \alpha$. Thus, a small α will decrease the size of the matrix and is flexible to change. The running time for this heuristic would be $O(|V|\alpha^3)$. In the next chapter, we explore the trade-off between the optimal conditional algorithm and this heuristic.

CHAPTER 6

EVALUATION

In this thesis, we evaluate the performance of the proposed preemption point placement methods over different kinds of randomly-generated control flowgraphs based on realistic parameters.

§Methodology. We randomly generate control flowgraphs satisfying the production rules in Figure 4.1. For our evaluation, we have fixed the number of basic blocks at 400 and the number of "high-level phases" of the flowgraph to be 30. Each high-level phase is either comprised entirely of sequential blocks or is a (non-nested) conditional block with a branching factor of two[1]. For each sequential block (i.e., either a sequential phase or one branch of the conditional), the number of basic blocks is uniformly generated from $[3, 10]$. In our experiments, we vary the number of paths from δ_s to δ_z by increasing the number of conditional blocks; thus, the number of paths is of the form 2^C where C is the number of conditional blocks in the generated control flowgraph.

In Bertogna et al. [6], the authors obtained CRPD and WCET costs for evaluation of their algorithm for sequential control flowgraphs by randomly generating parameters similar to the parameters of realistic code examples (e.g., Simulink code for aircraft and automotive control). In this chapter, we apply the same methodology to generate CRPD and WCET costs (i.e., $\xi(e_j)$ and $C(\delta_i)$, respectively). WCETs were generated according to a Gaussian distribution with a mean equal to 4000 nanoseconds and a variance of 3000 nanoseconds. The CRPD for each EPP is generated according to the method in Bertogna et al. [6] that correlates adjacent EPPs; we modify their method to account for conditional branching. Specifically, consider the preemption of $\xi(e_i)$ for EPP e_i:

[1]We do not nest in order to control the number of paths from δ_s to δ_z. We have observed that the results for nested conditional blocks are similar to the ones presented here.

$$\xi(e_i = (\delta_b, \delta_c)) = \sum_{\delta_a \in \Upsilon_i} \xi(\delta_a, \delta_b)/|\Upsilon_i| + \Delta_i \tag{6.1}$$

where $\Upsilon_i \stackrel{\text{def}}{=} \{\delta_a \in V : (\delta_a, \delta_b) \in E\}$, $\Delta_i \stackrel{\text{def}}{=} \mathsf{guas}(m_i, \sigma)$ and

$$m_i \stackrel{\text{def}}{=} \begin{cases} -M & \text{if } \sum_{\delta_a \in \Upsilon_i} \xi(\delta_a, \delta_b)/|\Upsilon_i| > \xi_{\max} \\ +M & \text{if } \sum_{\delta_a \in \Upsilon_i} \xi(\delta_a, \delta_b)/|\Upsilon_i| < \xi_{\min} \\ \mathsf{sgn}(\Delta_{i-1})M & \text{otherwise} \end{cases} \tag{6.2}$$

The variance σ quantifies the degree of variability between consecutive EPPs. We use the same parameters as Bertogna et al. [6]: $\sigma = 3000$, $M = 20$, $\xi_{\min} = 1000$, and $\xi_{\max} = 55000$.

In our experiments, we compare the optimal algorithm proposed in this paper described in Chapter 5.4 (denoted as "COND(OPT)") with an application of the sequential algorithm of Bertogna et al. [6] (denoted as "SEQ"). SEQ consists of generating each path in the conditional control flowgraph and applying the previously-proposed sequential algorithm to determine the EPPs along each path. The EPPs of each path are unioned together to obtain the overall EPP selection to conditional flowgraph. We also compare these algorithms with the heuristic proposed in Chapter 5.4 (denoted as "COND($\alpha \times \alpha$)" where a value of α equals 50, 100, 500). We execute the algorithms on a 2.4GHz 4-core Intel $i7$ machine with 6GB of RAM.

We vary the non-preemption window (denoted by Q in nanoseconds) and the number of conditional blocks in the graph (denoted by C, resulting in 2^C total paths from δ_s to δ_z). For each (Q, C) pair, we use the above methodology to generate 100 different control flowgraphs. For each control flowgraph, we apply both algorithms and measure the resulting WCET+CRPD and running time of the algorithm. The averages of the WCETs+CRPDs are plotted in Figures 6.1 and 6.3, and the average running algorithm

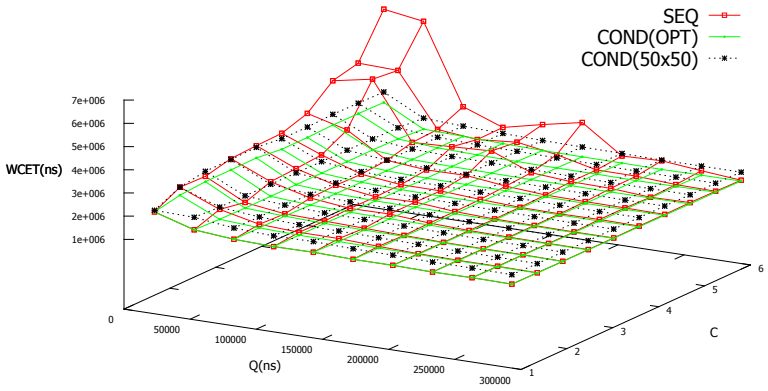


Figure 6.1: Comparison of WCET over Different Values of Q and Number of Conditional Blocks (C) for SEQ, COND(OPT), and COND(50×50).

running times in Figures 6.2 and 6.4.

§Results. Since COND(OPT) has been shown to be optimal for control flowgraphs considered in this paper, it is not surprising that Figure 6.1 shows that WCET of COND(OPT) never exceeds the WCET returned by SEQ or the heuristic COND(50×50). In Figure 6.2, as Q decreases, the preemption cost becomes a larger factor in the WCET since there are more preemptions due to the small non-preemptive window. As C increases, SEQ will add an increasingly large number of unnecessary EPPs to the solution set since it does not consider the conditional structure. These two observations explain the sharp increase in WCET for the sequential approach for small Q and C approaching a value of 6 (i.e., 2^6 total paths). After 2^6 paths, COND(OPT) memory requirements grow quickly for large Q, and the runtime of SEQ sharply increases. These increases make it difficult to scale SEQ or COND(OPT) to larger (Q, C) values.

In Figures 6.1 and 6.2, we observe that the heuristic COND(50×50) returns WCET close to the optimal with very small running time when compared with SEQ and COND(OPT). Thus, we further compare the various heuristics in terms of the accuracy/running-time tradeoff in Figures 6.3 and 6.4. These figures show that the accuracy does not change

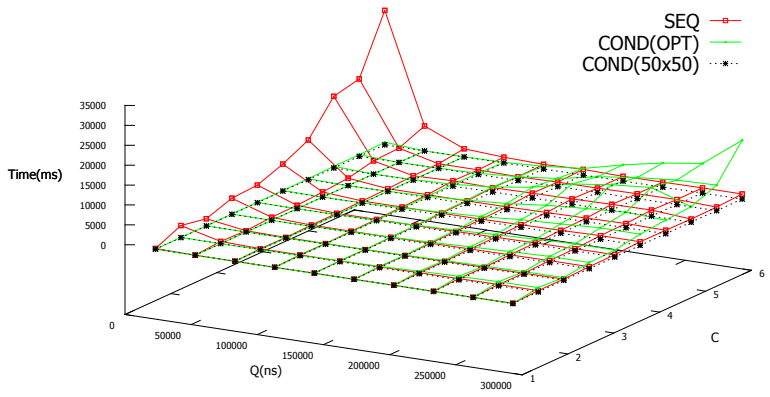


Figure 6.2: Comparison of Algorithm Running Times over Different Values of Q and Number of Conditional Blocks (C) for SEQ, COND(OPT), and COND(50×50)

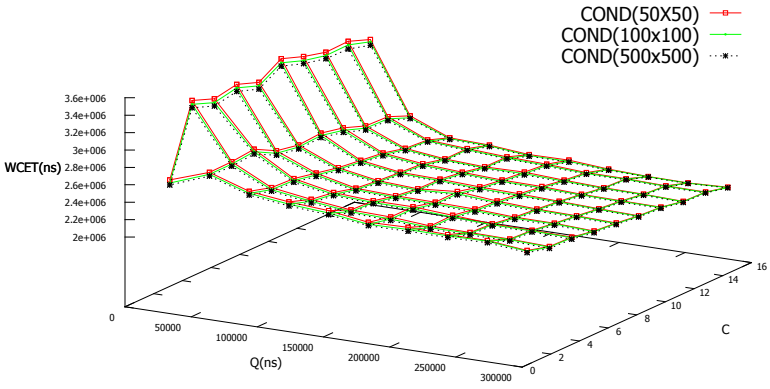


Figure 6.3: Comparison of WCET over Different Values of Q and Number of Conditional Blocks (C) for heuristics.

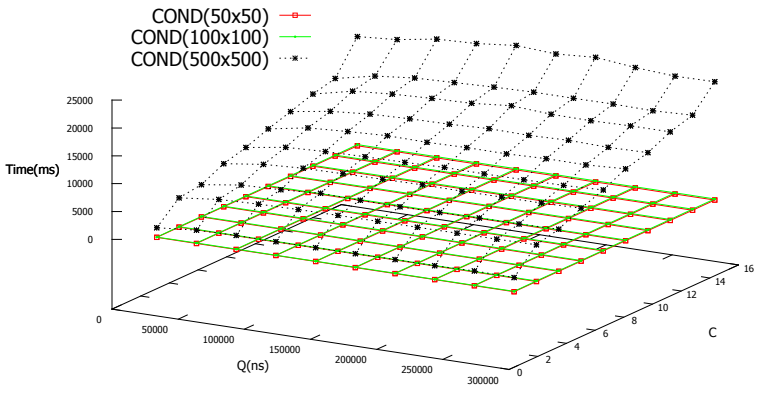


Figure 6.4: Comparison of Algorithm Running Times over Different Values of Q and Number of Conditional Blocks ((C) for heuristics.

much for α equal to 50, 100, or 500 for values of C up to 15 (i.e, there are up to 2^{15} paths in the graphs). However, for $\alpha = 500$, the memory costs of the cost matrices begin to dominate and increase the running time as the parse tree grows more complicated with increasing C. Thus, we can obtain a better accuracy-to-cost benefit with a relatively small value of α equal to 50.

CHAPTER 7

CONCLUSION & FUTURE WORK

In this thesis, we extended the applicability of existing techniques for the placement of preemption points to general tasks modeled with control flowgraphs, removing a pessimistic assumption that conditional blocks are contained within basic blocks. The proposed method allows for the optimal selection of the set of EPPs that minimize the resulting worst-case execution time, without affecting the schedulability of the system. The method uses a combination of graph grammars and dynamic programming, and runs in pseudo-polynomial time. Our evaluation shows that the algorithm can achieve a reduction in WCET+CRPD over approaches handling only sequential control flowgraphs. The improvement is particularly important when the allowed non-preemptive region length (Q) is small (i.e., for heavily loaded systems with limited slack). Such a system configuration is favorable also for the running time of the algorithm, which is $O(Q^3)$. We also have proposed heuristics which observe near-optimal behavior with very low runtime cost.

The future work is discussed as follows:

- Determine whether this is an NP-complete problem such that no polynomial-time algorithms exist. In this case, we will give an exact approximation scheme of the alternative heuristic algorithm. We will likely see that the implementation will return a better WCET+CRPD estimate given a fixed ε. Exact estimation of the use of memory will also help since the implementation runs out of bounds in a large Q.
- In planar separator theorem, an n-vertex planar graph can be split into smaller pieces by removing $\sqrt[2]{n}$ vertices. Each disjoint subgraph has at most $\frac{2n}{3}$ vertices. Using this theorem, the algorithms like approximation algorithm, divide and con-

quer, dynamic programming may yield a more efficient solution.

- Parameterized theory is a new paradigm which one can measure the time complexity of an algorithm not just in terms of the input length but also a small side parameter. Usually, the time complexity is linear in the input size and exponential in a function of the fixed parameter. Under this situation, a problem will have a better pseudo-polynomial time complexity when carefully choose the fixed parameter k. We would like to explore whether this problem is amenable to a fixed parameter algorithm.
- We also intend to explore the property of a "forward goto" statement, which means the code always runs a top-down mode and can not go backwards. An interesting question is whether we can solve the code containing a "forward goto" statement in pseudo-polynomial time. Intuitively, we think it is an NP-hard problem and will build a connection between the control flowgraph and the MAX-3-SAT problem. All in all, we will build a relationship among the edge weights, the value of Q and the WCET of a job. Then, we will transform to a MAX-3-SAT structure by assigning the boolean variables.
- We will study non-inline functions and non-unrolled loops in future work. The reasons for not unrolling loops and not inlining functions are sound: i) the program size will be decreased; and ii) subsequent invocations of the same instruction may be in the instruction cache, decreasing the CRPD. Therefore, it is important to be able to address programs with these substructures.
- As our approach utilizes tools from compilers, it would be interesting to see how this approach could be directly integrated into automatic programming tools to design real-time and embedded systems. aiT WCET Analyzer is a software that statically computes tight bounds for the worst-case execution time (WCET) of tasks in real-time systems. aiT WCET Analyzer directly analyzes binary executables and take the intrinsic cache and pipeline behavior into account. For certain target processors,

cache-related preemption costs can be taken into account. The input of the Analyzer can be high-level language like ANSI C. Therefore, doing the WCET optimization analysis on real code is interesting future work.

REFERENCES

[1] Antlrworks: The antlr gui development environment. http://www.antlr3.org/works/.

[2] A.V. Aho, R. Sethi, and J.D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, 1986.

[3] Sanjoy Baruah. The limited-preemption uniprocessor scheduling of sporadic task systems. In *Proc. of the 17th Euromicro Conf. on Real-Time Systems (ECRTS'05)*, pages 137–144, Palma de Mallorca, Balearic Islands, Spain, July 6-8, 2005.

[4] M. Bertogna and S. Baruah. Limited preemption EDF scheduling of sporadic task systems. *IEEE Transactions on Industrial Informatics*, 6(4):579–591, 2010.

[5] M. Bertogna, G. Buttazzo, M. Marinoni, G. Yao, F. Esposito, and M. Caccamo. Preemption points placement for sporadic task sets. In *Proceedings of the 22nd Euromicro Conference on Real-Time Systems (ECRTS'10)*, Brussels, Belgium, June 2010.

[6] M. Bertogna, O. Xhani, M. Marinoni, F. Esposito, and G. Buttazzo. Optimal selection of preemption points to minimize preemption overhead. In *Proceedings of the Euromicro Conference on Real-Time Systems*, Porto, Portugal, July 2011. IEEE Computer Society Press.

[7] Marko Bertogna, Giorgio Buttazzo, and Gang Yao. Improving feasibility of fixed priority tasks using non-preemptive regions. In *Proceedings of 32nd IEEE Real-Time Systems Symposium (RTSS 2011)*, Vienna, Austria, Nov. 30 - Dec. 2, 2011.

[8] Corrado Böhm and Giuseppe Jacopini. Flow diagrams, turing machines and languages with only two formation rules. *Commun. ACM*, 9(5):366–371, May 1966.

[9] R.J. Bril, J.J. Lukkien, and W.F.J. Verhaegh. Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption. *Real-Time System*, 42(1-3):63–119, 2009.

[10] Alan Burns. Preemptive priority based scheduling: An appropriate engineering approach. *S. Son, editor, Advances in Real-Time Systems*, pages 225–248, 1994.

[11] Alan Burns and Andy Wellings. *Real-Time Systems and Programming Languages*. Addison-Wesley, second edition edition, 1997.

[12] G. Buttazzo, M. Bertogna, and G.Yao. Limited preemptive scheduling for real-time systems: a survey. *IEEE Transactions on Industrial Informatics*, 9(1), 2013.

[13] Robert Davis and Marko Bertogna. Optimal fixed priority scheduling with deferred preemption. In *Real-Time Systems Symposium (RTSS 2012)*, San Juan, Puerto Rico, December 4-7, 2012.

[14] R. Gopalakrishnan and Gurudatta M. Parulkar. Bringing real-time scheduling theory and practice closer for multimedia computing. In *Proceedings of the 1996 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '96, pages 1–12, New York, NY, USA, 1996. ACM.

[15] Mathieu Grenier and Nicolas Navet. Fine-tuning mac-level protocols for optimized real-time qos, 1991.

[16] Ken Kennedy and Linda Zucconi. Applications of a graph grammar for program control flow analysis. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 72–85, New York, NY, USA, 1977. ACM.

[17] Sheayun Lee, Chang-Gun Lee, Minsuk Lee, SangLyul Min, and ChongSang Kim. Limited preemptible scheduling to embrace cache memory in real-time systems. In Frank Mueller and Azer Bestavros, editors, *Languages, Compilers, and Tools for*

Embedded Systems, volume 1474 of *Lecture Notes in Computer Science*, pages 51–64. Springer Berlin Heidelberg, 1998.

[18] A. K. Mok. *Fundamental Design Problems of Distributed Systems for The Hard-Real-Time Environment*. PhD thesis, Laboratory for Computer Science, Massachusetts Institute of Technology, 1983. Available as Technical Report No. MIT/LCS/TR-297.

[19] Aloysius K. Mok and Wing-Chi Poon. Non-preemptive robustness under reduced system load. In *Proceedings of the 26th IEEE International Real-Time Systems Symposium*, RTSS '05, pages 200–209, Washington, DC, USA, 2005. IEEE Computer Society.

[20] Moonju Park. Non-preemptive fixed priority scheduling of hard real-time periodic tasks. In Yong Shi, GeertDick Albada, Jack Dongarra, and PeterM.A. Sloot, editors, *Computational Science ICCS 2007*, volume 4490 of *Lecture Notes in Computer Science*, pages 881–888. Springer Berlin Heidelberg, 2007.

[21] Harini Ramaprasad and Frank Mueller. Tightening the bounds on feasible preemption points. In *IEEE Real-Time Systems Symposium*, pages 212–224, Rio de Janeiro, Brazil, 2006.

[22] Michael L. Scott. *Programming Language Pragmatics*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2009.

[23] J. Simonson and J.H. Patel. Use of preferred preemption points in cache-based real-time systems. In *Computer Performance and Dependability Symposium, 1995. Proceedings., International*, pages 316–325, Apr 1995.

[24] Yun Wang and M. Saksena. Scheduling fixed-priority tasks with preemption threshold. In *Real-Time Computing Systems and Applications, 1999. RTCSA '99. Sixth International Conference on*, pages 328–335, 1999.

[25] Reinhard Wilhelm et al. The worst-case execution-time problemoverview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 7(3):36:1–36:53, 2008.

[26] Y. Yesha X. He. Parallel recognition and decomposition of two terminal series parallel graphs. *Information and Computation*, 75(1):15–38, 1987.

[27] Gang Yao, Giorgio Buttazzo, and Marko Bertogna. Bounding the maximum length of non-preemptive regions under fixed priority scheduling. In *Proc. of the 15th IEEE Int. Conf. on Embedded and Real-Time Computing Systems and Applications (RTCSA 2009)*, Beijing, China, August 24-26, 2009.

ABSTRACT

EXPLICIT PREEMPTION PLACEMENT FOR REAL-TIME CONDITIONAL CODE VIA GRAPH GRAMMARS AND DYNAMIC PROGRAMMING

by

BO PENG

May 2014

Advisor: Dr.Nathan Fisher

Major: Computer Science

Degree: Master of Science

Traditional worst-case execution time (WCET) analysis must make very pessimistic assumptions regarding the cost of preemptions for a real-time job. For every potential preemption point, the analysis must add to the WCET of a job the *cache-related preemption delay* (CRPD) incurred due to the contention for memory resources with other jobs in the system. However, recent work has shown that CRPD can vary at each preemption point (due to the cache lines that must be reloaded for subsequent code after the preemption). Using this observation and information obtained from schedulability analysis on the maximum length of the non-preemptive region of a job, we seek to find the optimal set of explicit preemption-points (EPPs) that minimize the WCET and ensure system schedulability. Utilizing graph grammars and dynamic programming, we develop a pseudo-polynomial-time algorithm that is capable of analyzing jobs that can be represented by control flowgraphs with arbitrarily-nested conditional structures. This algorithm extends previous work that could only handle sequential flowgraphs. Exhaustive experiments are included to show that the proposed approach is able to significantly improve the bounds on the worst-case execution times of limited preemptive tasks.

AUTOBIOGRAPHICAL STATEMENT

Bo Peng

5200 Anthony Wayne Dr

Detroit, MI 48201

(313) 676-0218

Education:

Ph.D.	Computer Science
	Wayne State University, 2013-Present
	Advisor: Nathan Fisher
M.S.	Computer Science
	Wayne State University, 2013
	Advisor: Nathan Fisher
B.S.	Communication Engeerning
	Xidian University, 2011

Employment:

07/2013-Present:	Graduate Research Assistant, Wayne State University
12/2012-04/2013:	Graduate Student Assistant, Wayne Sate University.