Project no. 248828

ADVANCE

Strategic Research Partnership (STREP)
ASYNCHRONOUS AND **D**YNAMIC **V**IRTUALISATION THROUGH PERFORMANCE
ANALYSIS TO SUPPORT **C**ONCURRENCY **E**NGINEERING

Statistical Model of Resource Usage

D7

Due date of deliverable: January 31st, 2011
Actual submission date: March 11th, 2011

Start date of project: February 1st, 2010

Type: Deliverable
WP number: WP4
Task number: WP4a

Responsible institution: USTAN
Editor & and editor's address: Kevin Hammond
University of St Andrews
School of Computer Science
KY16 9SX St Andrews, Scotland

Version 1 / Last edited by Philip Hölzenspies / March 17, 2011

Project co-funded by the European Commission within the Seventh Framework Programme		
Dissemination Level		
PU	Public	√
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Revision history:

Version	Date	Authors	Institution	Section affected, comments
0.1	07/06/2010	Steffen Jost	USTAN	Initial version
0.2	01/03/2011	Philip Hölzenspies	USTAN	VR-Net-adaptions, project applications
1	11/03/2011	Philip Hölzenspies	USTAN	Copulas, composition, et al.

Tasks related to this deliverable:

Task No.	Task description	Partners involved°
WP4a	Develop statistical model of resource usage	HERTS, USTAN*, TWENTE

°This task list may not be equivalent to the list of partners contributing as authors to the deliverable
*Task leader

Executive Summary

The goal of work package 4 is to develop resource usage models and analyses that can be used to accurately predict execution time for VR-Net programs in terms of statistically valid throughput, latency and jitter metrics. The models and analyses must reflect on both user-supplied information, and on feedback from lower levels of the execution chain, including concrete hardware resources. They must be capable of producing statistically-valid resource usage information for heterogeneous multi-core/many-core architectures.

This task involves developing a statistically valid model of resource usage in terms of the key throughput, latency and jitter metrics for multi-core architectures. In order to do this, we will adapt existing theoretical models that capture WCET information to also represent probabilistic time information. We will develop new models that are capable of combining this base information in a rational way in order to yield statistically valid resource usage information, and will develop new models of memory allocation that can be used to derive high-quality information on throughput, latency and jitter.

Contents

Executive Summary 1

1 Introduction **4**
1.1 Methodology 5
1.2 Coordination language VR-Net 7
1.2.1 Asynchronous combinatorial stream programming . 7
1.3 A brief overview of VR-Net 8
1.3.1 Networks, records and streams 8
1.3.2 Types, type matching and routing 9
1.3.3 Flow inheritance 10
1.3.4 Primitive networks 11
1.3.5 VR-Net Network Combinators 13

2 Statistical Background **15**
2.1 Probability distributions 15
2.1.1 Discrete Probability Distributions 16
2.1.2 Continuous Probability Distributions 16
2.1.3 The Normal or Gaussian Distribution 17
2.1.4 The Uniform Distribution 18
2.1.5 Joint and Marginal Distributions 19
2.2 Relating Probability Distributions 20
2.2.1 Convolutions 20

3 Structural Model of Throughput, Latency and Jitter in terms of VR-Net Primitives **21**
3.1 Basics 21
3.2 Network composition rules 24

4 Using Copulas to Combine Throughput, Latency and Jitter Distributions **25**
4.1 Copulas 26
4.2 Copula Definitions 27
4.3 Combining Time Information using Copulas 28
4.3.1 Example use of Copulas 29

4.4 Applying copulas to composition 30
4.4.1 Example 30
4.5 Summary 31

5 Dealing with Queueing Issues **32**
5.1 Application 32
5.2 Execution model 33
5.3 Queueing 34

6 Outline Research Plan for WP4: Towards the Construction of a Statistically-Valid Automatic Analysis for VR-Net Programs **35**
6.1 Validation through Measurement Data 35
6.2 A Type-Based Inference System for VR-Net programs 36
6.3 Refinement to Virtual Hardware 37
6.4 Dealing with Queues 37

7 Conclusions **39**

Chapter 1

Introduction

This is the first deliverable for Work Package 4, representing the outcome of work performed in task WP4a. This task involved developing a statistically valid model of resource usage in terms of the key throughput, latency and jitter metrics for (heterogeneous) multi-core architectures. This represents a core deliverable both for WP4 and for the ADVANCE project as a whole. It will be built on in subsequent tasks both in this work package (WP4b and WP4c) and in the remainder of the project (notably in WP3 and WP5).

The overall vision for the ADVANCE project is shown in Figure 1.1. In essence, we adopt a statistically-informed approach to compiling for (possibly heterogeneous) multicore systems, using both user-provided information/constraints and dynamically obtained measurements. We synthesise code through a hardware virtualisation layer (SVP, as described in Deliverable D6). Deriving a statistical model of resource usage and properties of the temporal behaviour of a program, in order to match the ADVANCE project vision, is a two-phase activity. Static analysis of software components and their interconnection in component networks results in an initial statistical model. This model is concretized and adjusted by dynamic analysis, i.e. the analysis of measurements performed at run-time. Which measurements should be analysed and how is also determined in the static analysis.

This report details compile-time composition rules for VR-Net networks, that produce the aforementioned models. VR-Net is a coordination language, that abstracts away from application-specific values. It defines *streams* as sequences of *records*, where *records* are sets of named *fields* and *tags*. Tags have integer values, visible on the VR-Net-level, whereas the contents of fields are opaque. The set of names (or *labels*) of all tags and fields in a record together, is the type of the record. In VR-Net, *boxes* are stateless functions implemented in a programming language, mapping records of one type onto zero-or-more records of a specified set of output types. On the VR-Net-level, boxes are considered functions (or 'primitive networks')

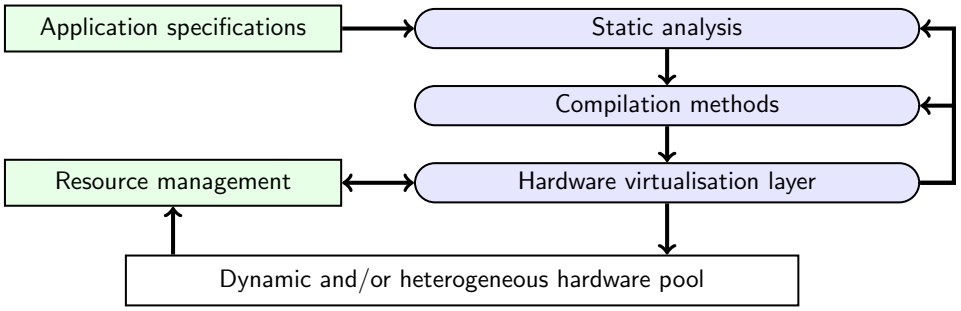


Figure 1.1: ADVANCE Project Vision

from a single input stream to a single output stream. Finally, VR-Net defines a set of combinators, with which networks (i.e. individual boxes or boxes already combined into non-primitive networks) may be combined. Every VR-Net-expression results in a network with one input stream and one output stream.

Because VR-Net networks may be defined for multiple *type variants*, the properties captured in the statistical model are defined per variant. The rules for the initial static analysis presented in this report assume fully typed arguments, i.e. these rules are applied in a stage *after* type inference and checking. It follows from this assumption, that propagating properties per type variant does not incur a combinatorical explosion.

1.1 Methodology

The purpose of our statistical analysis is to determine the overall execution costs for systems of components. In addition, we would like to know the execution durations of the individual components and their contribution to the final output latency.

We are interested in three key metrics: i) latency, i.e. how long a component takes before it responds to some input; ii) jitter, the variation in latencies; and iii) throughput, how much data can be processed by the system in some given time period. The key issue that we face is how to determine the behaviour of two or more components when they are combined into a larger system in terms of the individual behaviours of the underlying components. This behaviour may be given through user annotations, as described in Deliverable D4, or perhaps as determined automatically by analysis.

The ADVANCE approach combines measurement with statistically-based analysis as follows. We can easily obtain the throughput, latency and/or jitter of each component using straightforward measurements. Given a suf-

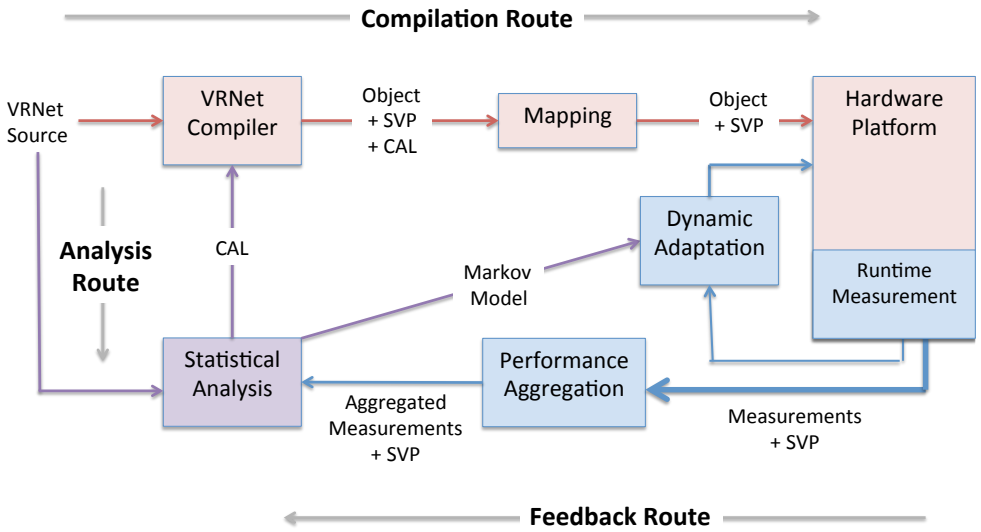


Figure 1.2: Overview of the ADVANCE Statistical Analysis Approach

ficient number of repeated runs, this can be used to set up the probability distribution of these metrics for the system as a whole, for the individual components and ideally for the joint latency distributions of each combined pair of components.

More detail on the approach taken by the ADVANCE project is outlined in Figure 1.2. (Possibly annotated) VR-Net code code describing the streaming system is provided to the ADVANCE system by the user. In the simplest form (shown in red on the top of the diagram), this VR-Net source may be compiled directly to object code, mapped to the available hardware (D5) using information provided through the SVP virtual hardware interface (D6), and then executed on the hardware platform. The more interesting approach involves, in addition, analysing the VR-Net source code using the analyses that will be developed in WP4, based on the models described in this deliverable, collating this information in the form of CAL annotations to the VR-Net source code that is provided to the VR-Net compiler and thus to the mapping software, and synthesising a (hidden) Markov model that can be used to take adaptive runtime decisions on dynamic placement, based on parameter values or other runtime measurement information. The analysis route is shown in purple along the left and centre of the diagram. The statistical analysis is informed by runtime measurements that must be aggregated to provide the probability distributions that the analysis is based on. This route is shown in blue along the bottom of the diagram.

1.2 Coordination language VR-Net

In this section, we discuss concepts of VR-Net, an asynchronous stream coordination language. These concepts are required for asynchronous combinatorial stream programming.

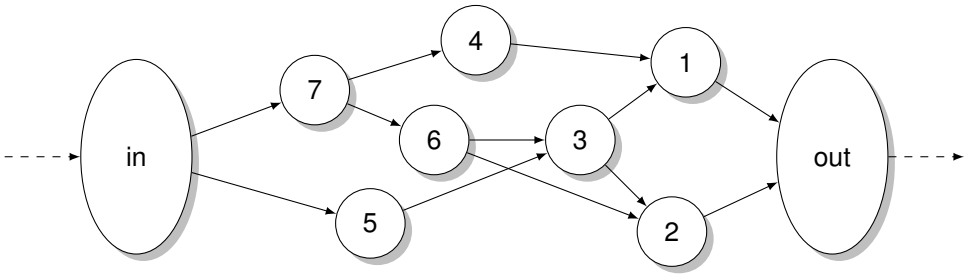
1.2.1 Asynchronous combinatorial stream programming

Networked stream programming goes back to Kahn's networks [10] which are fixed graphs with message streams flowing along the edges and stream-processing functions placed at the vertices. The importance of this type of computing is in its simple fixed-point semantics and the static nature of task distribution (discussed above). It is due to these characteristics that networked stream programming is used widely in control systems (for example the Airbus software [3] is written in a stream processing language ESTEREL [2]). However, with the advent of multicore systems and especially large, heterogeneous, many-/multicore architectures, the synchrony found in most programming tools of this kind will become more and more of a limiting factor for throughput and utilization maximization. Consequently asynchronous stream-processing languages, such as VR-Net [6] are likely to prove to be useful. The principles behind asynchronous stream-processing can be found in [16]; here we only restate some ideas required to understand the work presented in this thesis.

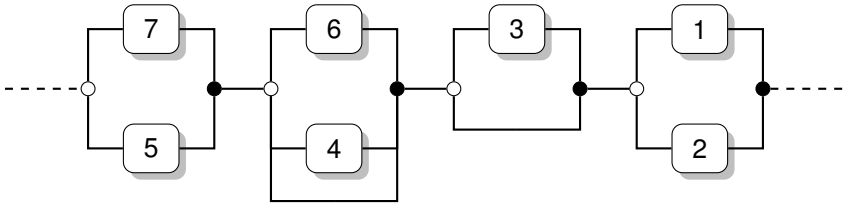
1.2.1.1 MIMO vs. SISO

Figure 1.3(a) shows an arbitrary streaming network, where vertices are functions of multiple streams producing multiple streams (the top diagram). This is referred to as Multiple In, Multiple Out (MIMO). For simplicity, the network is assumed to be acyclic. The input stream α is split by the vertex *In* into streams carrying messages that are intended for specific input ports of individual vertices. The output of the graph is gathered by the vertex *Out* into a single output stream. Assuming that the vertices can respond to the input messages on different ports irrespective of their mutual timing (the assumption of asynchrony), multiple input streams to a vertex can be merged into one, where the messages themselves are labelled with the port information. Similarly multiple output streams could be labelled and merged in such a way. Thus, any asynchronous MIMO network can be rewritten to a Single In, Single Out (SISO) network. The example network is rewritten in 1.3(b).

In the rewritten network, the black bullets are non-deterministic stream mergers and the circles are splitters. The position of a vertex in the rewritten network is determined by the longest path to that vertex from the network input in the original network. Bypasses (identity functions) are added



(a) MIMO network



(b) SISO network

Figure 1.3: A MIMO network and its equivalent SISO network

when a vertex requires messages from not-immediately-preceding stages. The topology of SISO networks can be constructed with algebraic expressions, with networks as operands and combinators as operators. Any (valid) expression of this form is again a SISO network. Two combinators are used in Figure 1.3(b); serial and parallel composition. VR-Net provides more than these two combinators; they are described in section 1.3.

1.3 A brief overview of VR-Net

In this section, we introduce essential features of the language VR-Net, a declarative coordination language for asynchronous stream programming, to serve as background information. We deliberately exclude key language features, like type inference, that are not directly related to the subject of the work presented in this thesis. Moreover, the syntax used here is enriched with some shorthand notations, with regards to the official VR-Net syntax. For a complete coverage of VR-Net, including the official syntax, we refer the interested reader to [6].

1.3.1 Networks, records and streams

Every network in VR-Net is SISO. This means that every network transforms an input stream to an output stream. A stream is a (potentially infinite) sequence of non-overlapping, discrete data items, called *records*. The

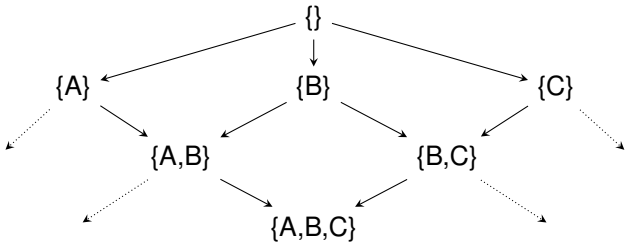


Figure 1.4: Subtype relation between record types

basic networks are *primitive networks* that can be combined by using *network combinators* into (non-primitive, SISO) networks.

Primitive networks perform either *processing* or *synchronization*. Processing networks are stateless functions, defined by the user in one of two possible ways: A *box*, implemented in a programming language (referred to as the *box language*), or a *filter*, specified in VR-Net terms. Synchronization networks, known as *synchroqueues*, combine records based on their type. All three forms of primitive networks are described in more detail in seciton 1.3.4.

Records contain *fields* and *tags*. Both of these are named. Fields contain values that are opaque to VR-Net, i.e. they are values for the box language. VR-Net can only rename, copy or delete fields, but any other transformation on fields occurs in boxes only. Tags contain integer values and are readable and writeable in VR-Net.

1.3.2 Types, type matching and routing

Records are routed through a network dependent on their type. A *record type* is specified by the set of names of the fields and tags contained in records of that type. A subtype relation on record types is defined as the superset relation on sets, i.e. when record type t contains strictly more names than record type t', then t is a subtype of t'. Figure 1.4 illustrates the subtype relation. A, B and C are the names of fields or tags. The universal supertype is the empty set. Every set with precisely one name is a subtype of the empty set. Every set with two names is a subtype of both sets that contain only one of those names. The subtype relation is transitive, e.g. (from the example in Figure 1.4) $\{A, B\}$ is a subtype of $\{B\}$, which is a subtype of $\{\}$, so $\{A, B\}$ is a subtype of $\{\}$ also.

A network takes records from its input stream and results in records on its output stream. Thus, *network types* are defined in terms of record types. Networks take records of one type and result in zero-or-more records of possibly different types. Networks can take different types of records as

input. These are referred to as *input variants*. The types of the records on a network's output stream depend on the input variant and (often) also on the *values* of fields and tags in the corresponding records on the network's input stream. Thus, for every input variant, a set of *output variants* (record types of records that the network can produce in response to the input) is given. Formally, if $\mathfrak{R}$ denotes the set of all possible record types, the set of all network types (denoted $\mathfrak{N}$) is defined as $\mathfrak{N} = \mathscr{P}(\mathfrak{R} \times \mathscr{P}\mathfrak{R})$, where $\mathscr{P}$ denotes the power set. However, for the work presented in this thesis, only the input type of a network (the set of all the network's input variants) is important, because these types are used for the routing of records within the networks. For a detailed discussion of the VR-Net type system and inference of network types, see [4].

If records can be used as input for a number of different networks—e.g. in case of parallel composition, discussed below—records are routed into the network with the *strongest matching type*. Here, a record type t_r matches a network type t_n when t_n contains at least one input variant t_v that is equal to or a supertype (i.e. subset) of t_r. This means that a network does not need all fields and tags in a record. In the next section, we discuss how the fields and tags not handled by the network are taken into account. The *strength* with which t_r matches input variant t_v is the size of t_v, i.e. the number of names in t_v. The strength with which t_r matches network type t_n is that of the strongest matching input variant of t_n.

1.3.3 Flow inheritance

As mentioned, records can be of a subtype of an input variant of a network. In such a case, the network does not affect the fields and tags with names not in the input variant. The fields and tags that are not indicated in the input variant of the network are said to 'flow around' the network. This is known formally as *flow inheritance*.

For example, consider a network with only one input variant $\{A, B\}$ and a record with type $\{A, B, C, D\}$. The record can be fed into the network, because its type is a subtype of the network's input variant. However, because the network is only defined to work on the fields (or tags) with the names A and B, only those fields are inserted. In other words, the record is split into a *through*- and an *around*-part. The through-part is a record $\{A, B\}$ and the around-part is a record with type $\{C, D\}$. The through-part is inserted into the network, in response to which a result is produced. The around-part is combined with the result. If the result is a record of type $\{X, Y\}$, then the combination of the result and the around-part is a record of type $\{C, D, X, Y\}$. When the result contains fields with names that also occur in the around-part, the fields in the result are preferred. Thus, if the result is a record of type $\{C, X\}$, the combined record has the type $\{C, D, X\}$ and the value of the field labelled C is that of field

C from the result of the network.

Flow inheritance is only defined on *primitive* networks. Intuitively, many non-primitive networks behave as if flow inheritance was defined on them as well. However, such an intuitive understanding is often very misleading. The primitive networks and their behaviour with regards to flow inheritance is discussed in seciton 1.3.4. Section 1.3.5 describes the network combinators. For a more elaborate discussion on flow inheritance and its consequences on combined networks, see [4].

1.3.4 Primitive networks

As discussed above, VR-Net distinguishes three different forms of primitive networks: boxes, filters and synchroqueues. The primitive networks are discussed in more detail in this section.

1.3.4.1 Box

An VR-Net box is a stateless user-defined processing primitive network. It is connected to the outside world via a single input stream and a single output stream. A box can start processing as soon as a record appears on its input stream. The concrete behaviour of the box is specified outside VR-Net using an appropriate *box language*. A box may emit zero or more records on the output stream in response to a record on the input stream; the exact number depends both on the box' implementation and the values of the incoming record. We call this dynamic behaviour the *multiplicity* of the box.

Every execution of a box takes exactly one record from the input, i.e. multiplicity only occurs on the box' output. The entirety of a box' output after consuming a single record is called the box' *response* to that record. Furthermore, since the box is stateless, the response of the box depends exclusively on the single record of the input. Because a box is applied in-order to the records on its input stream, it carries over the order of the records on its input stream into a *causal order* on the output stream. In other words, causal order means, that if record a precedes record b on the output stream, one of two cases hold: Either the two are part of the same response to a record on the input stream *or* the record on the input stream to which a was (part of) the response preceded the record to which b was (part of) the response.

Flow inheritance for boxes with multiplicity is defined as follows. The around-part of an incoming record is combined (as described above) with *each* record in the box' response to the through-part. For example, assume a box that accepts records of type $\{A, B\}$ and produces records either of type $\{X, Y\}$ or of type $\{C, X\}$. Now assume an input record with type $\{A, B, C, D\}$, which is a subtype of the box' only input variant. The input

record is split into a through-part, having type $\{A, B\}$ and an around-part, having type $\{C, D\}$. The through-part is consumed by the box. In response, the box produces two records: r of type $\{X, Y\}$ and r' of type $\{C, X\}$. The around-part is combined with both results. For r this results in a record of type $\{A, B, X, Y\}$. However, r' contains a field or tag with the same name C is a field or tag in the around-part. Thus, combining r' with the around-part, discards the C field or tag from the latter. In other words, C and X are taken from r' and D from the around-part to come to the combined result with type $\{C, D, X\}$.

1.3.4.2 Filter

A filter is similar to a box in terms of its behaviour as a processing primitive network on a stream, including causal order, statelessness, multiplicity and flow inheritance. It is different, because it is specified in VR-Net by means of (simple) expressions and it can not access fields (i.e. box language values).

1.3.4.3 Synchroqueue

Synchroqueues in VR-Net are infinite sequences of *synchrocells*. The purpose of synchrocells is to combine two or more records into a single record, based on their respective types. A synchrocell is defined by a list of record types. For each record type, there is a corresponding 'slot' in the synchrocell. In every slot, one record of that type can be stored. When a single record is stored in a slot, that slot is *filled*. Before that time, the slot is *free*.

Like boxes and filters, a synchrocell takes records from its input stream in-order and one at a time. Every incoming record is matched against those record types that correspond to free slots. For every record type of the synchrocell that the incoming record matches, the part of the incoming record corresponding to the record type of the synchrocell is stored in the corresponding slot, i.e. the through-part of the record with regards to the record type is stored. When an incoming record fills all remaining free slots, the synchrocell *syncs*: The records stored in the slots of the synchrocell are combined into a single record. This combined record is produced on the output.

This leads to three distinguishable scenarios that describe the synchrocell's behaviour: 1) If the incoming record matches the record types of all remaining free slots, the record resulting from the combination of all stored records is produced on the output. 2) If the incoming record matches at least one record type corresponding to a free slot (but not all remaining), nothing is produced. 3) The incoming record is produced 'as is' when it does not match any remaining free slot's record type. Thus, multiplicity occurs also for synchrocells: The response to a record is either one record or nothing. Finally, the slots of the synchrocell are *not* freed when the synchro-

cell syncs. This implies that, after syncing, a synchrocell passes through all incoming records (scenario 3).

Flow inheritance on synchrocells is defined slightly differently than on boxes and filters: The around-part of the (first) record that matches the first record type (in order of the specification) of the synchrocell is merged with the result when the synchrocell syncs. The around-parts of the other records are discarded. This behaviour is an VR-Net design choice.

Being infinite sequences of synchrocells, synchroqueues synchronise entire streams. In other words, a synchroqueue combines the n^{th} occurrence of every record type in its definition and produces the combined result as its n^{th} output.

1.3.5 VR-Net Network Combinators

VR-Net primitive networks, i.e. boxes, filters and synchroqueues, are combined into (acyclic, SISO) streaming networks by means of network combinators. In other words, network construction is an inductive process starting with boxes, filters and synchroqueues as basis. In the following, different network combinators are explained.

1.3.5.1 Sequential composition

Given two networks N and M, the sequential composition of N and M (denoted N .. M) yields a network where all input is streamed into N, the output stream of N becomes the input stream of M and the output stream of M becomes the output stream of the combined network.

In principle, the sequential composition is similar to function composition. Sequential composition naturally preserves the causal order.

1.3.5.2 Parallel composition

Given two networks N and M, the parallel composition of N and M (denoted $N \parallel M$, or $N \mid M$) is a network where the input stream is split up into two streams that are fed into N and M, respectively. More precisely, records on the input stream are routed to the network with the strongest matching type. If there are multiple networks with equally strong matching types, a non-deterministic choice is made among those networks. The individual output streams of N and M are merged to form the output stream of the new network. The merger is either deterministic ($N \parallel M$) or non-deterministic ($N \mid M$).

Non-deterministically merging two streams means that records on those streams are non-deterministically *interleaved*. However, when record a appears before record b in the output stream of N, a also appears before b in

the merged output stream. The non-deterministic merger of the two output streams of N and M does not preserve the causal order on the stream, since data being processed by N may well be overtaken by data processed by M or vice versa. The deterministic merger *does* preserve the causal order along both substreams.

1.3.5.3 Repetition composition

The repetition composition of a network N means, that every record in the output stream of N may be fed through N again, until it matches a *terminator*. A terminator is a set of *patterns* (see [6]). A pattern consists either of a single record type to be matched or of a record type and a *guard expression*. Guard expressions are expressions of tag values. Guard expressions have a boolean result. A record matches a pattern if it matches the pattern's record type and, if the pattern has a guard expression, the pattern's guard expression evaluates to *true*. A record matches a terminator if it matches any of the terminator's patterns.

Given a network N and a terminator γ, $N \setminus \gamma$ denotes the repetition composition of N, which is itself a SISO network. We can interpret γ as a function on records, i.e. $\gamma : \mathcal{R} \rightarrow \{iterate, done\}$, where $\mathcal{R}$ denotes the set of all possible records. Thus, the application of γ to a record a, i.e. $\gamma(a)$, is the matching of a against the patterns of γ. If any pattern matches, the result of this function is $done$, otherwise, it is $iterate$.

Repetition composition preserves the causal order of the input stream in the overall output stream of the repeatedly applied network, whereas the non-deterministic variant does not (necessarily).

1.3.5.4 Inspection composition

Given a network N, a boolean value δ and the name of a tag t, the inspection composition of N inspecting t (denoted N !! t or N ! t) is a potentially infinite parallel composition of N with itself, where incoming records are routed based on the *value* of tag t. In other words, for every value of tag t, a separate instance of N exists to which all records that hold the corresponding tag value are routed. This composition can be thought of as an equivalent to the switch-/case-statement found in many programming languages. As before, there are two variants: deterministic (N !! t) and non-deterministic (N ! t).

Chapter 2

Statistical Background

This chapter provides some basic background in statistics that underlies the technologies that we are deploying in the ADVANCE project. In particular, we discuss probability distributions, including the Gaussian or normal distribution, joint and marginal distributions and how probability distributions can be related using convolutions.

2.1 Probability distributions

Much of the work described in this deliverable is based around determining the probability that some metric has a particular value, and in combining those probabilities in some rational way. In order to achieve this, it is necessary to understand the underlying probability distribution functions. A *probability distribution* characterises the set of probabilities that we are considering over some range of "random" variables. We use a *probability density function* to describe the distribution. A distribution may be either *discrete*, ranging over a set of discrete argument values, such as the integers, or *continuous*, ranging over a set of continuous argument values, such as the real numbers. In the work that follows, we will mainly be dealing with timing infomation, in the form of throughput, latency and jitter. We will therefore generally be dealing with *continuous* distributions. For completeness and clarity, we will however give a brief introduction to both discrete and continuous probability distributions here.

2.1.1 Discrete Probability Distributions

The mathematical definition of a *discrete probability function*, $p(x)$, is a function that satisfies the following properties.

a) The probability that x can take a specific value is $p(x)$. That is

$$P[X = x] \;=\; p(x) = p_x.$$

b) $p(x)$ is non-negative for all real x.

c) The sum of $p(x)$ over all possible values of x is 1, that is

$$\sum_j p_j \;=\; 1.$$

where j represents all possible values that x can have and p_j is the probability at x_j.

One consequence of the second and third properties is that $0 \leq p(x) \leq 1$. Since the values are discrete, the condition that the probabilities sum to one means that at least one of the possible values has to occur.

2.1.2 Continuous Probability Distributions

The mathematical definition of a *continuous probability density function*, $f(x)$, is a function that satisfies the following properties.

a) The probability that x lies between two points a and b is:

$$p[a \leq x \leq b] = \int_a^b f(x)\,\mathrm{d}x.$$

b) This probability is non-negative for all real x.

c) The integral of the probability function is one, that is

$$\int f(x)\,\mathrm{d}x \;\;=\;\; 1.$$

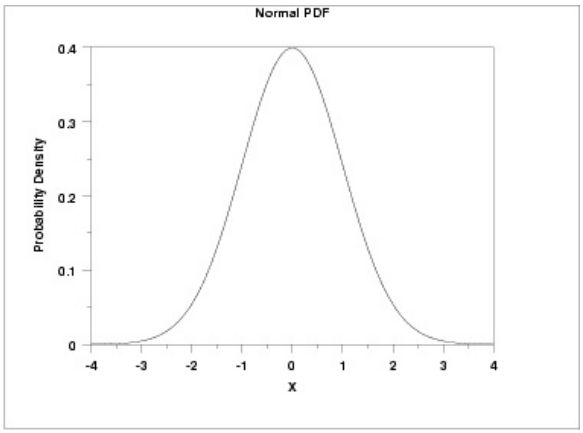
The area under the curve between two distinct points defines the probability for that interval. The third property, using the integral, is equivalent to the property for discrete distributions that the sum of all the probabilities must equal one.

2.1.3 The Normal or Gaussian Distribution

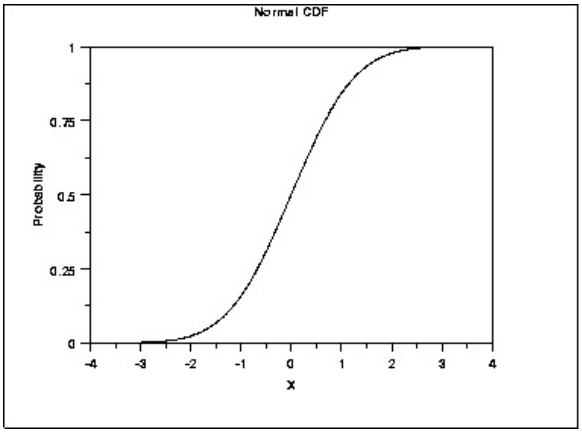
The *normal distribution*, or *Gaussian distribution* is probably the most important distribution in statistics. Many standard statistical tests assume that the data follow a normal distribution and the normal distribution is also used to find significance levels in many hypothesis tests and confidence intervals. The general formula for the probability density function of the normal distribution is

$$f(x) = \frac{e^{-(x-\mu)^2/(2\sigma^2)}}{\sigma\sqrt{2\pi}}$$

where μ is the *location parameter* and σ is the *scale parameter*. The location and scale parameters of the normal distribution can be estimated using the values of the sample mean and sample standard deviation, respectively. A normal distribution can be completely characterised by its mean and standard deviation. The case where $\mu = 0$ and $\sigma = 1$ is called the standard normal distribution. The plot of the standard normal probability density function is shown below. This follows the familiar classical bell-curve shape.



There is no simple closed formula for the *cumulative distribution function* of the normal distribution. Rather, it is computed numerically. The plot of the standard normal cumulative distribution function is shown below.



The normal distribution is widely used, partly because it is both well behaved and mathematically tractable. However, the *central limit theorem* provides a theoretical basis for why it has wide applicability. The central limit theorem essentially states that as the sample size (N) becomes large, the following occur:

a) The sampling distribution of the mean becomes approximately normal regardless of the distribution of the original variable.

b) The sampling distribution of the mean is centered at the population mean, μ, of the original variable. In addition, the standard deviation of the sampling distribution of the mean approaches $\sigma/\sqrt{N}$.

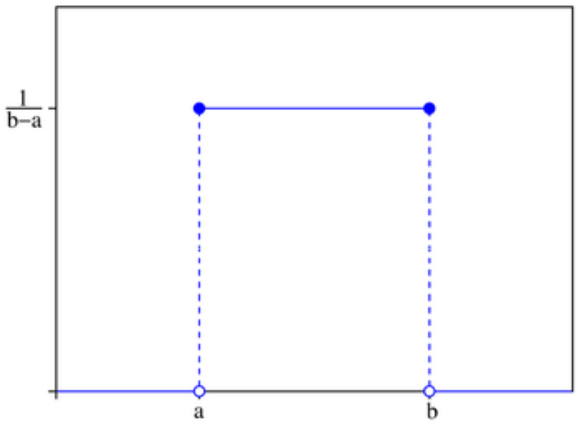
We will use the Gaussian distribution as a simple underpinning for the statistical model of VR-Net primitives that we will develop in Chapter 3.

2.1.4 The Uniform Distribution

A continuous *uniform distribution* (or *rectangular distribution*) occurs where all outcomes within a specified range are equally probable. The general formula for the probability density function of the continuous uniform distribution is

$$f(x) = \begin{cases} \frac{1}{b-a} & a \leq x \leq b \\ 0 & \text{otherwise} \end{cases}$$

The plot of the uniform probability density function for some interval a, b $(U(a,b))$ is shown below.



The cumulative distribution function gives a straight line from 0 to 1.0 between a and b. We exploit uniform distributions in Chapter 4.

2.1.5 Joint and Marginal Distributions

The *joint distribution* for two random variables X and Y defines the probability of those events that require both X and Y. For continuous distributions, given the joint probability density function $f(x, y)$ over values in X and Y, we can determine the overall joint distribution.

$$P\{(X, Y) \in \mathbb{R}\} = \int \int_R f(x, y)\, \mathrm{d}x\, \mathrm{d}y$$

Given a collection of two or more random variables comprising some joint probability distribution, the *marginal distribution* of a subset of those variables is the probability distribution of the variables in that subset. The marginal probability distribution functions for X and Y, f_X and f_Y, are defined as:

$$f_X(x) = \int_{-\infty}^{\infty} f(x, y)\mathrm{d}y$$
$$f_Y(y) = \int_{-\infty}^{\infty} f(x, y)\mathrm{d}x$$

2.2 Relating Probability Distributions

When all probability distributions are Gaussian, they can be easily related by calculating joint means and standard deviations. For example, for random variables X and Y, whose Gaussian distributions are defined by (m_X, σ_X) and (m_Y, σ_Y), respectively, we can calculate the sum of the distributions by:

$$\begin{aligned} m_{X+Y} &= m_X + m_Y \\ \sigma_{X+Y} &= \sqrt{\sigma_X^2 + \sigma_Y^2 + 2\rho\sigma_X\sigma_Y} \end{aligned}$$

where ρ denotes the *correlation* between X and Y.

2.2.1 Convolutions

When summing two uncorrelated random variables which do not have Gaussian distributions, then the distribution of their sum is given by the *convolution* of the two component distributions*. In mathematics, a convolution is used to express the overlap of two functions. The convolution $(f * g)(t)$ of two functions f and g over a finite range $[0, t]$ is given by:

$$(f * g)\,(t) = \int_0 (f(T).g(t-T))\mathrm{d}T$$

where $*$ represents the convolution operator. Convolutions naturally lend themselves to computational solutions. For example, we can use a convolution to compute the joint probability distribution of the two marginal distribution functions f_X and f_Y up to some point i, as follows.

$$\mathrm{conv}_{XY}\,(i) = \sum_{j=0}^{i} f_X(j).f_Y(i-j)$$

*Chapter 4 will consider situations when even convolutions are inadequate, and we may need to use a *copula*.

Chapter 3

Structural Model of Throughput, Latency and Jitter in terms of VR-Net Primitives

In order to obtain a good understanding of the problem, we will devise an initial statistical model based on strong simplifying assumptions. The statistical basis for the model is discussed in Chapters 2 & 4 and some problems with this initial model are discussed in Chapter 5.

3.1 Basics

We formulate our statistical analysis for a very simple subset of VR-Net first, where we make the following simplifying assumptions, in order to obtain an initial working system.

Definition 3.1 (Notions)**.** Basic VR-Net properties to be determined by analysis.

Latency execution time of a single box or network, in response to a single input record;

Jitter variance in latency, i.e. variation of network execution time;

Throughput number of records a network can consume over a finite time interval.

Definition 3.2 (Single Token VR-Net)**.** We formulate our rules on a simplified VR-Net, where we assume the following:

a) Fully typed, mapped to singular type. Every network is fully typed, including the parts of the type imposed by other networks with which

the network is composed. In other words, there is no implicit flow-inheritence in the types with which the networks are annotated. The types in fully typed networks can be mapped down to singular types, since they are only required for (local) routing decisions. We assume such singleton types where for types $a \neq b$ it holds that a and b are completely independent.

b) No multiplicity. For each input record, each box produces precisely one output record. This assumption excludes the modelling of the behaviour of synchroqueues. The implications of the behaviour of synchroqueues and specifics of the VR-Net execution model are discussed in Chapter 5.

c) No data-dependent types. Networks can produce output of only one specified type in response to an input record. However, for different types of input records, differently typed output records may be produced.

d) Input labels matter. For each box and for each set of input labels, we are given the latency distribution. This is not a distribution over the input values themselves, but just of time required to process them.

e) Field values do not matter significantly. This means that the runtime of a box can be sufficiently described in relation to the provided input labels, but is largely independent from the actual values supplied. This assumption implies, that data-dependent implications for the temporal behaviour of a box should be made explicit in the record type.

Definition 3.3 (Annotated Types)**.** We augment the standard VR-Net types with probability annotations for properties, i.e. we add stochastic variables to the type signatures that are distributed according to some probability density function. In any implementation, the (representations of) probability density functions are used to represent these variables. A type T_A for a VR-Net network A consists of an unordered set of tuples, written

$$\Gamma \vdash A : T_A$$
$$T_A = \{\langle\langle a_1, b_1\rangle, \delta_1\rangle, \ldots, \langle\langle a_n, b_n\rangle, \delta_n\rangle\}$$

where a_i, c_i are singleton types and δ_i are stochastic variables representing the properties we want to derive. These sets can be seen as relations, relating paths through network A that a record with type a_i may follow to result in a record of type c_i to the value of the property represented by δ_i. It is a (finite partial) relation, rather than a (finite partial) function, because there may be multiple paths from some type p to some type q, related to different variables, i.e. some $i \neq j$ where $\langle a_i, c_i\rangle = \langle a_j, c_j\rangle$.

On these types, we define three more operators with the following notation. The *domain* of type T_A (denoted $\mathrm{dom}(T_A)$) is the set of all type-pairs, that occur as a left argument in the relation T_A. The *range* of type T_A (denoted $\mathrm{ran}(T_A)$) is the set of all variables, that occur as a right argument in the relation T_A. The *domain restriction* of type T_A to the set of type-pairs D (denoted $T_A \upharpoonright D$) is the largest subrelation of T_A that is defined only for left arguments that occur in D. Thus, for T_A as used above:

$$\begin{aligned}\mathrm{dom}(T_A) &= \{\langle a_1, b_1\rangle, \ldots, \langle a_n, b_n\rangle\} \\ \mathrm{ran}(T_A) &= \{\delta_1, \ldots, \delta_n\} \\ T_A \upharpoonright D &= \{\langle\langle a, b\rangle, \delta\rangle \mid \langle\langle a, b\rangle, \delta\rangle \in T_A \wedge \langle a, b\rangle \in D\}\end{aligned}$$

The meaning of the annotated VR-Net type given above is that, for example, the latency of box A follows the probability distribution δ_i if the box is executed on an input record of type a_i. Note that by our assumptions (Def. 3.2), the type of the input record largely determines the latency of the box, with the actual values contained in the record being of little significance with respect to the latency. Hence, a single probability distribution per triple suffices to model latency. The same argument holds for throughput and jitter.

When the output of one network is fed into another, the paths through the combination of networks must be captured in the combined types. We describe composition rules in terms of VR-Net-combinators in the next section. However, these rules require a transitive combinator on types, that constructs the new type with all paths through a sequentially combined couple of networks.

Definition 3.4 (Transitive type combination)**.** Given two annotated network types T_A, T_B, the *transitive type combination* $T_A \otimes T_B$ is defined as

$$T_A \otimes T_B = \mathrm{flatten}\{\langle\langle a, d\rangle, \delta_A \oplus \delta_B\rangle \mid \langle\langle a, b\rangle, \delta_A\rangle \in T_A \wedge \langle\langle c, d\rangle, \delta_B\rangle \wedge b = c\}$$

where the concrete definitions of both $\mathrm{flatten}$ and $\oplus$ depend on the property being propagated (discussed further in Chapter 4). However, they can be understood generally by their use. $T_A \oplus T_B$ expresses the sequential composition of the behaviours T_A and T_B, such that records following the path to which this combination is associated incur first T_A and then T_B. It follows that $\oplus$ is *not* generally commutative (i.e. not for every property). Because there may be multiple paths through a network with the same from- and to-type and since different paths typically incur different δs, one type pair may be related to a set of variables, rather than just one. The function $\mathrm{flatten}$ converts the relation into a function by combining the different probability distribution functions related to the same type-pairs into one.

3.2 Network composition rules

In this section, we demonstrate how the annotated types of networks are composed in the construction of networks.

Serial Composition .. Serial composition combines the types of the networks under composition in a transitive fashion.

$$\frac{\Gamma \vdash A{:}T_A \qquad \Gamma \vdash B{:}T_B}{\Gamma \vdash A \mathbin{..} B{:}T_A \otimes T_B} \text{ (SERIAL COMPOSITION)}$$

Parallel Composition $\|$ Parallel composition implies, per record, a choice between either alternative. Paths through the network are not combined through parallel composition, but remain independent.

$$\frac{\Gamma \vdash A{:}T_A \qquad \Gamma \vdash B{:}T_B}{\Gamma \vdash A \parallel B{:}\,\mathrm{flatten}(T_A \cup T_B)} \text{ (PARALLEL COMPOSITION)}$$

Repetition Composition $\setminus$ Because we are combining probability distribution functions, the number of repetitions is not a simple constant with which the aggregated property can be multiplied. Instead, we must combine n instances of the same probability distribution function with itself, when there are n repetitions. Since the number of repetitions is not known at this point, we choose a free variable to represent the number of repetitions a record undergoes and propagate this variable outward. Ultimately, this n is represented in a CAL-expression (Deliverable D4).

$$\frac{\Gamma \vdash A{:}T_A \qquad \gamma \subseteq \mathrm{ran}(\mathrm{dom}(T_A)) \qquad \mathrm{free}(n)}{\Gamma, n \vdash A \setminus \gamma{:} \bigotimes_{i=1}^{n} T_A} \text{ (REPETITION COMPOSITION)}$$

Inspection Composition ! Treatment should be largely identical to parallel composition.

Synchroqueues $[\,|\,]^\star$ Since synchroqueues do not contain any code a static distribution depending on the machine and memory models should be used.

These composition rules are illustrated with an example, after the discussion of $\mathrm{flatten}$ and $\oplus$, in Chapter 4.

Chapter 4

Using Copulas to Combine Throughput, Latency and Jitter Distributions

A key question that we face in determining the statistically valid combination of component costs is how to relate the probability distributions for the underlying components. Given two boxes B_X and B_Y, with random variables X and Y representing the observed values of throughput, latency or jitter for each box, we need to determine the combination of these variables $Z = X \oplus Y$. We need to cover the four basic cases where:

a) X and Y are independent;

b) the dependencies between X and Y are known and given by a *joint distribution*, $J(z) = P[X \oplus Y] \leq z$;

c) the dependencies between X and Y are partially known; and

d) there is some dependency between X and Y, but this is not known.

Much statistical theory is based around an independence assumption, corresponding to the first case above. In this case, we may be able to combine similar distributions as we did above for the Gaussian distribution or by using an appropriate convolution. In the second case, where we have completely measured the *joint distribution* of the combined system, this can be used directly as a predictor of overall behaviour. The most difficult situations arise where dependencies are partially or completely unknown. Fortunately, we can obtain some information about the dependencies through careful measurement of box behaviours. The difficulty is to extrapolate from these measurements in order to determine the overall behaviour of a system.

4.1 Copulas

An approach that has recently drawn considerable attention, especially in financial modelling, is the use of *copulas* [14, 5]. A copula is a *joint distribution function* that has uniform marginals. In effect, copulas are functions that allow the modelling of joint distributions and their dependencies on the basis of just the marginal probability densities of the single components. For example, given measured latencies for components and some combination of those components, we can then calculate a copula that relates the full joint probability distribution function to those for the components as a function over the original distribution functions. Given known costs for the components, we can use this copula to predict the overall cost for the combination of the components.

The idea of copulas has previously been deployed for worst case execution time analysis [1], where it has been used to predict probabilistic bounds on execution times for specific execution paths under known input values. The main novelties that we will achieve in the ADVANCE project are:

a) applying the idea of copulas to stream-based programs, including parallel execution and other VR-Net constructs;

b) abstracting to network-level constructs, rather than expression-level constructs;

c) deploying this idea to also cover average-case behaviour in terms of throughput, latency and jitter;

d) using it as the basis for a type-based cost inference; and

e) applying it in the context of a feedback-directed compilation and execution environment for heterogeneous multicore systems.

We anticipate that the use of copulas will actually be more useful in our setting than their use by Bernat et al., since

a) we do not need to guarantee worst-case behaviours, in badly behaved situations, we can use non-limit copulas to cover average-case behaviours rather than using the *supremal* copula, as Bernat et al. do;

b) there will be relatively few VR-Net boxes in a network, which means the computational cost for combining distributions should be more tractable than for Bernat et al.; and

c) we have a rational means of separating sub-structural behaviours.

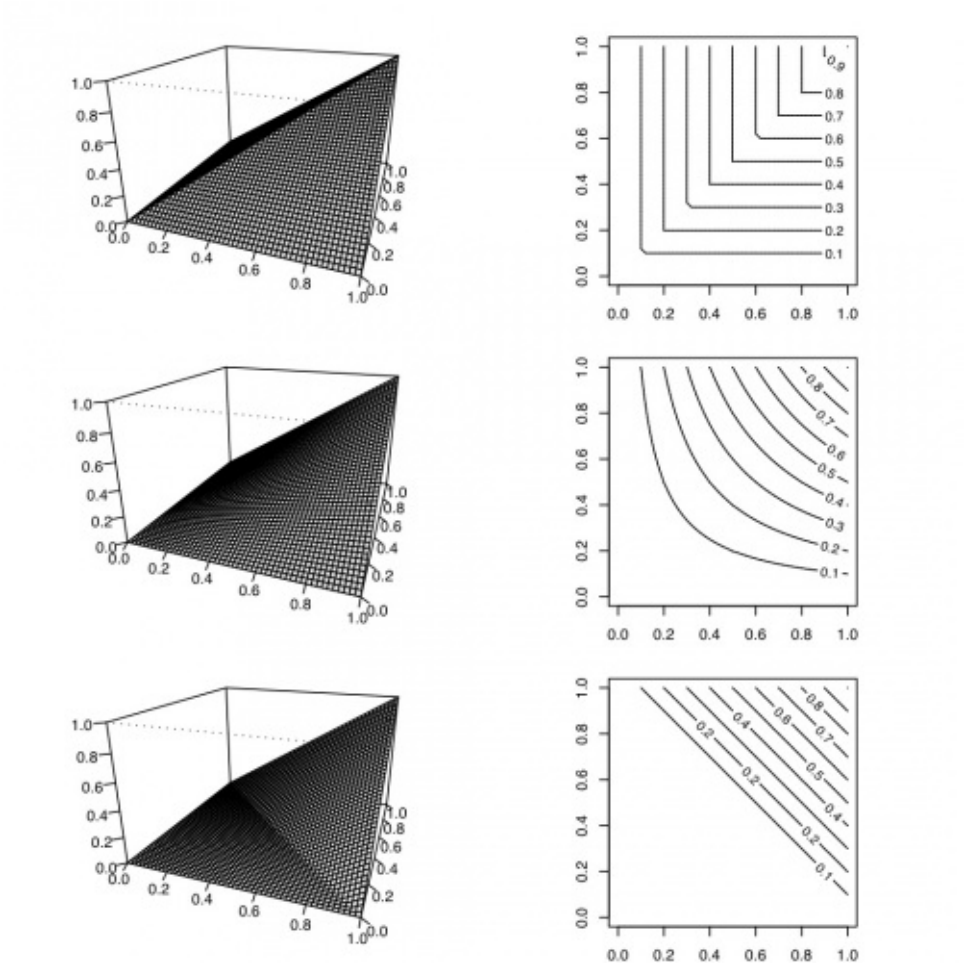


Figure 4.1: Plots of the Freéchet W (above), M (below) and Π copulas

Finally, unlike the approach taken by Bernat et al., which synthesises a program to determine overall system behaviour in terms of a combination of copulas over marginal distributions, we use a more rigorous type-based approach that allows us to systematically combine information from components, and will build an automatic analysis for some given composition of components.

4.2 Copula Definitions

A *copula* is a function $C(x, y)$ that satisfies the following properties:

$$C(x,0) = C(0,y) = 0$$
$$C(x,1) = x$$
$$C(1,y) = y$$

It must also satisfy the *monotonicity* property:

$$\begin{array}{ll} \text{If} & 0 \leq x_1 \leq x_2 \text{ and } 0 \leq y_1 \leq y_2, \\ \text{then} & C(x_1,y_1) - C(x_1,y_2) - C(x_2,y_1) + C(x_2,y_2) \geq 0 \end{array}$$

Given a joint distribution H with continuous marginals F and G, there exists a *unique* copula C, such that for all $x, y \in \mathbb{R}^*$,

$$H(x,y) \;=\; C(F(x),G(y)) \quad [\text{ Sklar}'\text{s Theorem [15] }]$$

Copulas have a number of useful properties. In particular, there exist standard lower and upper bound copulas (the Fréchet W and M copulas) that bound all valid copulas:

$$W(u,v) \leq C(u,v) \leq M(u,v)$$

The W copula represents complete negative dependence between variables, and the M copula represents complete positive dependence. Plots of the Fréchet W and M copulas are shown in Figure 4.1. The W copula is shown above and the M copula is shown below. The central plot is of the product Π copula, also known as the *independence copula*, a copula where there is no dependence anywhere between variables.

$$H(x,y) = F(x) \cdot G(y)$$

While we have considered only the summing the property here (since this is all we require to combine the distributions of latency and jitter, that we use in this deliverable), copulas can be generalised to instantiate $\oplus$ to arbitrary non-decreasing and continuous combination functions, including $+$, $\times$ and *max*. They can also easily be extended to the multivariate case.

4.3 Combining Time Information using Copulas

We will focus here on the key metric of latency and its associated jitter, based on measurements of time delays. Dealing with throughput also requires us to consider concurrency issues. Given a probability distribution for time delays, we can use this to estimate values for latency and jitter as the first and second *moments* of the distribution, that is as the population mean and variance.

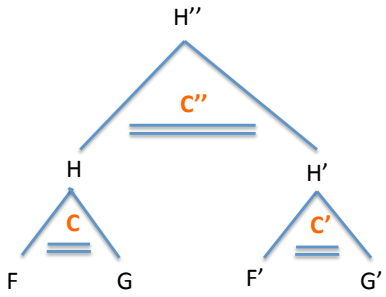


Figure 4.2: Relating Probability Distribution Functions (F, G, H etc.) using Copulas (C, C' etc.)

We need to obtain estimates of the latency and jitter for a VR-Net network that is composed from several sub-networks using the binary operators described above. Statistically, the main problem that we face is therefore one of determining a probability distribution for the combination of two random variables X and Y when their dependencies may be partially or completely unknown, where X and Y represent possibly-dependent distributions on time delays for different sub-networks. That is we need to determine the joint probability distribution function for $Z = X \oplus Y$, representing the time delay for the combined network, given probability distribution functions for X and Y. Fortunately, copulas provide a way to do this. The behaviour of the combination of the two variables X and Y is completely defined given the marginal distributions for each of the variables and a copula. The copula yields the joint probability distribution for the combination given the marginal distributions for X and Y. For any given joint probability distribution, this copula is *unique* if the variables are continuous. The copula describes the dependence structure, and the marginals describe the observable behaviour of the individual components. We can thus separate our problem into one of observation (for the marginals) and calculation (to determine the dependence structure).

4.3.1 Example use of Copulas

For example, given the VR-Net network

$$(B_X \,..\, B_Y) \,..\, (B_{X'} \,..\, B_{Y'})$$

our approach is as follows: given the two marginal probability distributions, F and G obtained from measurements for B_X and B_Y and some known joint distribution H over their combination $Z = X \oplus Y$, we can

construct the uniform marginals and the corresponding copula C. The copula therefore relates the two marginal distributions. We can do the same for a different set of marginal distributions F' and G', representing the serial composition of the second set of boxes. Now, we must extrapolate this to the complete VR-Net network. Since H and H' are themselves simply probability distribution functions, we can apply the same principle, ending up with a final copula C'' that relates these intermediate distributions to the joint distribution H''. This is shown in Figure 4.2. The final joint distribution H'' over time delays can then be used to estimate latency and jitter for the system as a whole. Having obtained this set of copulas from measurement training data, we can now use them to predict the results for some live input values or set of input values. Over time, as we obtain more measurement results, the copulas can be recalculated using this measured data and so improved to reflect increased knowledge about the performance of the application.

4.4 Applying copulas to composition

As discussed above, the $\oplus$ operator for any two variables is defined as a copula representing the combination of the corresponding property. Since $\mathrm{flatten}$ is to combine probability distributions for independent paths through the network, it employs the Π-copula, viz.

$$\mathrm{flatten}(T) = \{\langle t, \Pi\,(\mathrm{ran}(T \rhd \{t\}))\rangle \mid t \in \mathrm{dom}(T)\}$$

4.4.1 Example

Given boxes A, B, C, D and network N where

$$\begin{aligned} N &= (A \mathbin{..} B) \mathbin{..} (C \parallel D) \\ \Gamma &\vdash A{:}\{a \xrightarrow{\delta_A} b\} \\ \Gamma &\vdash B{:}\{b \xrightarrow{\delta_{Bc}} c, b \xrightarrow{\delta_{Bd}} d\} \\ \Gamma &\vdash C{:}\{c \xrightarrow{\delta_C} e\} \\ \Gamma &\vdash D{:}\{d \xrightarrow{\delta_D} e\} \end{aligned}$$

then we can construct the annotated type for N by composition, using the rules defined above.

Deriving for $A \mathbin{..} B$

$$\frac{\Gamma \vdash A{:}\{a \xrightarrow{\delta_A} b\} \qquad \Gamma \vdash B{:}\{b \xrightarrow{\delta_{Bc}} c, b \xrightarrow{\delta_{Bd}} d\} \qquad \delta_{ABc} = \delta_A \oplus \delta_{Bc} \qquad \delta_{ABd} = \delta_A \oplus \delta_{Bd}}{\Gamma \vdash A..B{:}\{a \xrightarrow{\delta_{ABc}} c, a \xrightarrow{\delta_{ABd}} d\}} \quad (\textsc{Ser})$$

and for $C \parallel D$

$$\frac{\Gamma \vdash C{:}\{c \xrightarrow{\delta_C} e\} \qquad \Gamma \vdash D{:}\{d \xrightarrow{\delta_D} e\}}{\begin{array}{c}\Gamma \vdash C \parallel D{:}\,\mathrm{flatten}\{c \xrightarrow{\delta_C} e, d \xrightarrow{\delta_D} e\} \\ (\textit{evaluation of } \mathrm{flatten}) \\ \Gamma \vdash C \parallel D{:}\{c \xrightarrow{\delta_C} e, d \xrightarrow{\delta_D} e\}\end{array}} \quad \text{(PAR)}$$

allowing us to derive for N

$$\frac{\begin{array}{c}\Gamma \vdash A..B{:}\{a \xrightarrow{\delta_{ABc}} c, a \xrightarrow{\delta_{ABd}} d\} \qquad \Gamma \vdash C \parallel D{:}\{c \xrightarrow{\delta_C} e, d \xrightarrow{\delta_D} e\} \\ \delta_{N1} = \delta_{ABc} \oplus \delta_C \qquad \delta_{N2} = \delta_{ABd} \oplus \delta_D\end{array}}{\begin{array}{c}\Gamma \vdash N{:}\,\mathrm{flatten}\{a \xrightarrow{\delta_{N1}} e, a \xrightarrow{\delta_{N2}} e\} \\ (\textit{evaluation of } \mathrm{flatten}) \\ \Gamma \vdash N{:}\{a \xrightarrow{\Pi\{\delta_{N1}, \delta_{N2}\}} e\}\end{array}} \quad \text{(SER)}$$

Thus, the probability of a record of type a incurring a latency x through network N is

$$P(x) = (\delta_A \oplus \delta_{Bc} \oplus \delta_C)(x) \cdot (\delta_A \oplus \delta_{Bd} \oplus \delta_D)(x)$$

4.5 Summary

Copulas form a promising approach to the problem of combining information from multivariate probability distribution functions, separating dependence structure from the underlying distributions. They have been widely applied in a number of areas, including, rather notoriously, finance. Used properly, they form a valuable tool for understanding and reasoning about stochastic dependence. Mikosch has, however, raised a number of concerns [12, 13] about their indiscriminate use (especially for some kinds of financial modelling), especially where particular families of copulas are used without understanding their underlying properties, or where predictions are made on the basis of false assumptions or without considering dynamic changes to the underlying probability distributions. By constructing specific copulas for particular combinations of distributions and by periodically reconstructing the copulas in the light of improved information about execution costs, we believe that these valid concerns may be met when modelling and predicting the behaviours of VR-Net boxes as part of the ADVANCE approach. Their predictions must, of course, be carefully tested against real VR-Net networks and revised in the light of experience. There is also one important limitation that we will consider in the next chapter: copulas are not capable, by themselves, of dealing with stochastic queueing behaviours. In order to deal with such behaviours, we will need to consider additional techniques, such as (hidden) Markov models.

Chapter 5

Dealing with Queueing Issues

This chapter describes some issues with the rules presented in Chapter 3. Firstly, the problems arising from one of the ADVANCE applications, taken from our commercial partners at Philips Healthcare, are discussed. Secondly, there is a large class of execution models for VR-Net under which the temporal behaviour of a running program is not captured by our rules. This class and the problem it raises for our rules are discussed. Finally, we discuss the commonality between these problems.

5.1 Application

In a medical imaging application under study, a stream of images needs to be analysed and displayed. For display purposed, the images need to undergo noise reduction. Part of the analysis involves feature detection. Since this feature detection can be performed on the image without noise reduction, these two tasks can happen in parallel. Features found in the image must be displayed to the user. Therefore, the result of the noise reduction and the feature detection need to be combined somewhere.

A rough initial description of this program can already be given. Let N, F be the noise filter box and feature detector box, respectively. Furthermore, let $\Gamma \vdash N{:}\{\{v\} \xrightarrow{\tau_N} n\}$ and $\Gamma \vdash F{:}\{\{a\} \xrightarrow{\tau_F} f\}$. The network to implement the behaviour described above is as follows:

$$[\{i\} \to \{v = i\}, \{a = i\}]..(N \mid F)..[\mid n, f \mid]^\star$$

The VR-Net-filter duplicates all records with a field with label i into records with labels v (visualisation) and a (analysis). Each duplicate is routed to its corresponding branch in the parallel composition of N and F. Finally, the synchroqueue combines the corresponding results. Aside from the fact that the rules have not yet been generalised to cover multiplicity, additional constraints from the application cause a more fundamental problem.

Because the application is to be used during surgical procedures, there is a strict upper-bound to the acceptable latency per image, i.e. indicently, an image may violate the per-image deadline, but this may not impose significant delays on all consecutive images. In the above snippet of the full application, the feature detection is the time-critical part and the part that demonstrates the most erratic temporal behaviour. When the feature detection box violates its (soft) deadline, the delay imposed on the next images must be compensated for by not performing feature detection for the next image. However, since the synchroqueue requires a feature detection result for every noise reduction result, a dummy result must be produced for the feature detection.

The implementation details of this in VR-Net-terms is beyond the scope of this document. However, it should be clear from this example that when image i takes too long in feature detection, the run-time of the feature detection box for image $i + 1$ becomes c for a given (and small) constant c, i.e. the time to produce a dummy result. This time-dependent inter-image dependency can not be expressed by the rules presented in chapter 3. By simply taking c as the execution time of F for image $i + 1$, the run-time perceived distribution of exection times of F gets such a large varience, that it may lose its informative value.

5.2 Execution model

If a VR-Net-program is implemented with an execution model that interprets a box as a thread with a bounded input buffer and unbuffered output, another problem arises. The time a record spends in the input buffer is not modelled explicitly in our rules. By itself, this could be compensated by modelling additional wait-time as a (throughput) jitter. Although the rules do not currently derive either throuput or jitter, they can be adapted to propagate those properties as well. However, the parallel composition causes a problem that can not be solved by such straightforward additions.

In the network $A..(B \mid C)$, the input buffers of both B and C impose a blocking conditon on A, i.e. when the buffer of the network that A's result is routed to is full, A can not produce its result. Until A has produced its result, A does not proceed to process records in its input buffer. Thus, the throughput model of B is a dependency for the latency model of C and vice versa, but only when buffers ever cause backpressure.

Again, simply accepting the backpressure as an increasing factor of A's execution time takes away prohibitively from the informative value of A's models.

5.3 Queueing

The common denominator in both problems discussed above is inter-record dependency and the lack of a means to predict the types of future (or past) records. Expanding the rules to include the continuation for every network and taking into account the waiting time imposed by a continuation may compensate somewhat for these problems. However, this still does not provide extra information about the types of records in the stream. The derivation of a temporal model for the paralel composition will therefore remain either a sizable over-estimation or prohibitively uninformative.

Chapter 6

Outline Research Plan for WP4: Towards the Construction of a Statistically-Valid Automatic Analysis for VR-Net Programs

The workplan for Workpackage 4 aims at the development of an automatic statistically-valid static analysis for VR-Net programs that is capable of predicting key throughput, latency and jitter metrics. It also aims at the validation of this analysis against the applications that will be developed in WP7. The work reported above forms a good basis for the remainder of the work that must be carried out in the context of the ADVANCE project. Firstly, we must validate the theoretical models that we have developed above against concrete measurement information, obtained from real hardware and applications. Secondly, we must define and build the static analysis itself to calculate probability distribution functions for networks of VR-Net boxes. Thirdly, we need to refine the approach so that it deals properly with the ADVANCE virtual hardware abstractions. Fourthly, we need to deal with queueing issues, as identified in Chapter 5. Finally, we need to expose information in the form of CAL annotations (Deliverable D4) to guide the compilation process and Markov models to guide dynamic adaptivity.

6.1 Validation through Measurement Data

The models that we have developed in Chapters 3-4 need to be fully grounded in measurements of VR-Net networks. We intend to do this by populating probability distribution functions for real VR-Net programs on real hardware. We have undertaken some preliminary work to determine empirically the behaviours that are possible for compositions of VR-Net boxes.

We need to extend those measurements to cover the recently developed exemplar applications from Philips Healthcare, BioID, SCCH and SAP These empirical results must then be compared against the results that are predicted by the type rules from Chapter 3 in order to determine the accuracy and effectiveness of the predictions that we have obtained. This validation and measurement needs to be carried out on heterogeneous multicore platforms.

6.2 A Type-Based Inference System for VR-Net programs

The structural rules that we gave in Section 3.2 form the basis for a *type system* for systems of VR-Net boxes. We have previously applied similar systems as part of a type-based *amortised analysis* that can determine bounds on worst-case execution times for programs regardless of their input values [9, 8]. Our previous approach gives a *compositional* analysis, where costs for program expressions are determined by combining costs for subexpressions. It is straightforward to adapt this approach to the VR-Net setting: there are no new technical problems, the main difficulties arise from the use of stochastic information, and the need to use statistical combinators such as convolutions or copulas. Each of the structural rules gives rise to one or more inference rules that can be used to form the basis of a practical static analysis. For example, we can define rules for simple boxes and for serial composition as follows:

$$\frac{\Gamma(A) \;=\; \{a \xrightarrow{\pi_A} b\}}{\Gamma \vdash box \;\; A : \{a \xrightarrow{\tau_A} b\} \;\mid\; \{\tau = \pi_A\}} \qquad \text{(BOX)}$$

$$\frac{\Gamma \vdash A{:}\{a \xrightarrow{\tau_1} b\} \;\mid\; \Psi_A \qquad \Gamma \vdash B{:}\{b \xrightarrow{\tau_2} c\} \;\mid\; \Psi_B}{\Gamma \vdash A..B : \{a \xrightarrow{\tau} c\} \;\mid\; \Psi_A \cup \Psi_B \cup \{\tau = \tau_1 \oplus \tau_2\}} \text{(SERIAL COMPOSITION)}$$

Each VR-Net construct is evaluated in the context of an environment Γ that defines the type and associated probability distribution for each box in the VR-Net program. The result of each rule yields a set of *constraints*, Ψ_A etc, that govern the relationships between the type terms. Once the inference engine has inferred the set of constraints for an VR-Net program, they may be solved externally and their results associated with the corresponding type term. The rules above show that τ is associated with the probability distribution π_A for the Box rule, and expose the $\oplus$ operator on the underlying distributions for the Serial Composition rule. This approach has the advantage of separating the inference engine from the solver, allowing us to use well-understood type inference techniques to infer the structure of the solution, and simple, external solver techniques on the constraints that

are generated, while maintaining the link between the constraints and the types through the type variables τ etc. In previous work, we have also needed to define substructural rules to allow, for example, the weakening of cost information that is produced when combining information from structural rules for sub-terms in rules such as that for serial composition above. Since we are providing exact operators for combining distributions, such as convolutions or copulas, we do not believe that weakening rules will be necessary.

Since there is a close correlation between the rules describing the analysis and the structural rules of Section 3.2, It would be straightforward to prove their completeness. Proving their soundness involves defining an operational semantics for VR-Net box compositions and demonstrating that the results produced by the inference system is consistent with this semantics. We do not, however, anticipate doing this work as part of the ADVANCE project, since it would be time consuming and is not technically necessary for the production of a practical analysis.

6.3 Refinement to Virtual Hardware

The analysis rules that we have discussed above assume that probability distributions may be freely combined. As part of the ADVANCE project, It will be necessary to include information about virtual hardware abstractions, as defined in Deliverable D6. This will allow the mapping of boxes to alternative hardware on the basis of dynamically obtained performance or other information. The information about virtual hardware affects the base probability distribution information (so that a single box or combination of boxes may have several alternative probability distributions associated with it, reflecting possible hardware mappings), and perhaps the precise convolutions or copulasf, but will not affect the structure of the analysis. This means that, while some recomputation may be needed, the bulk of the analysis will remain unchanged, in line with the unchanged structure of the VR-Net program.

6.4 Dealing with Queues

While the copula approach seems promising for composing information about combinations of VR-Net boxes where we do not need to consider queues, since queues must be made explicit for at least one project application, a stream-centric (as opposed to box-centric) approach must also be explored. We propose to layer the two approaches, so that we can freely compose systems with or without queueing requirements. Types of predecessor and successor boxes have a crucial influence on queueing behaviour, since they are used to determine the routing of records through networks.

Any stream-centric model must, therefore, be type-aware, i.e. it must predict the occurrence of records of certain types.

Of the possible approaches, work on deriving Markov models [11] seems most promising to achieve this. Instead of composing statistical models for the extra-functional behaviours of networks as for the non-queued case, (hidden) Markov models may be used to model the "state" and "behaviour" of a stream in the presence of a queue. Since all boxes are Single In, Single Out, a behavioural description of a box can be used to derive a (hidden) Markov model for the output stream, given a (hidden) Markov model for the input stream. In the case of operators that split the input stream (e.g. parallel composition) can explicitly derive models for the streams fed into the networks being composed, based on the expected types in the input stream and the input types of the networks.

Chapter 7

Conclusions

We have provided a set of structural rules describing how statistical cost information in the form of probability distributions may be obtained for VR-Net programs, and systematically combined to yield statistically-valid combinations of cost infomation. We have shown how to define appropriate combining operators both in the simple case where we are combining independent Gaussian distributions, and in the more complex cases where we may not have precise information about dependencies. In order to do this, we have introduced the idea of a *copula*. This seems a promising approach for dealing with situations where we do not need to account for queuing. We have outlined how we will now take this work forward in constructing a statistically-valid analysis that can be used to predict the behaviour of VR-Net programs on (heterogeneous) multicore architectures, and we have also considered the key characteristics of the applications described by the industrial partners in the ADVANCE project.

Alternative execution models, where records are threads [7, Ch.7], only incur queueing latencies in the (partial) reordering required for in-order synchronisation in synchroqueues. In networks where paths between consecutive synchronisation points can be enumerated, these paths can be modelled with a ruleset such as described in this deliverable. In this case, continuations would not need to be modelled explicitly. Queueing analysis would still be required for every synchronisation point in the network, but any dependency between them could then be described by a single statistical model, resulting from this compositional approach. The feasibility of this method is subject to future research, however.

Bibliography

[1] Guillem Bernat, Alan Burns, and Martin Newby. Probabilistic timing analysis: An approach using copulas. *J. Embedded Comput.*, 1(2):179–194, 2005.

[2] Gérard Berry and Georges Gonthier. The ESTEREL synchronous programming language: design, semantics, implementation. *Science of Computer Programming*, 19(2):87–152, November 1992.

[3] J Binder. Safety-critical software for aerospace systems. *Aerospace America*, pages 26–27, August 2004.

[4] Haoxan Cai, Susan Eisenbach, Alex Shafarenko, and Clemens Grelck. Extending the S-Net Type System. In *Proceedings of the Æther-Morpheus Workshop From Reconfigurable to Self-Adaptive Computing (AMWAS'07), Paris, France*, 2007.

[5] Manuel Davy and Arnaud Doucet. Copulas: A new insight into positive time-frequency distributions, 2003.

[6] Clemens Grelck, Sven-Bodo Scholz, and Alex Shafarenko. A Gentle Introduction to S-Net: Typed Stream Processing and Declarative Coordination of Asynchronous Components. *Parallel Processing Letters*, 18(2):221–237, 2008.

[7] P. K. F. Hölzenspies. *On run-time exploitation of concurrency*. PhD thesis, Univ. of Twente, Enschede, The Netherlands, April 2010.

[8] S. Jost. *Linearly Bounded Heap Space Analysis, Ludwig-Maximilians-Universität, München, Germany (in preparation)*. PhD thesis, 2010.

[9] Steffen Jost, Hans-Wolfgang Loidl, Kevin Hammond, and Martin Hofmann. Static determination of quantitative resource usage for higher-order programs. In *POPL '10: Proceedings of the 37th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 223–236, New York, NY, USA, January 2010. ACM.

[10] G Kahn. The semantics of a simple language for parallel programming. In L Rosenfeld, editor, *Information Processing 74, Proc. IFIP Congress 74. August 5-10, Stockholm, Sweden*, pages 471–475. North-Holland, 1974.

[11] S.P. Meyn. *Markov Chains and Stochastic Stability (Communications & Control)*. Springer-Verlag New York Inc., 1993.

[12] Thomas Mikosch. Copulas: Tales and facts, 2005. http://www.math.ku.dk/~mikosch/Preprint/Copula/s.pdf.

[13] Thomas Mikosch. Copulas: Tales and facts – rejoinder, 2006. http://www.math.ku.dk/~mikosch/Preprint/Copula/answer.pdf.

[14] Roger B. Nelsen. *An Introduction to Copulas, 2nd Edition*. Springer, 2006.

[15] A. Sklar. Fonctions de répartition à n-dimensions et leurs marges. *Publications de l'Institut de Statistique de L'Université de Paris 8*, pages 229–231, 1959.

[16] SNet. Snet declarative coordination language, 2008.