



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Leandro Rafael Moreira Gomes

Weighted computations: semantics and program logics

Programa de Doutoramento em Informática
das Universidades do Minho, de Aveiro e do Porto

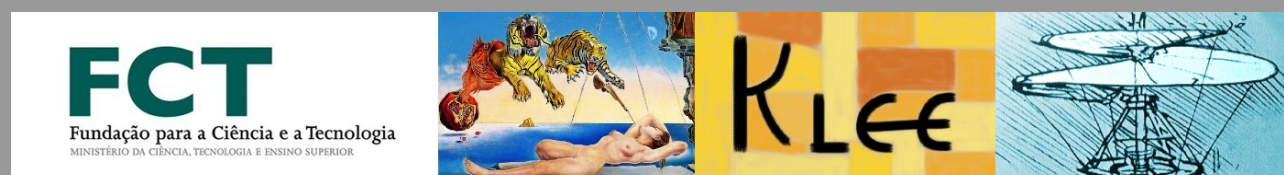


Universidade do Minho

Weighted computations: semantics and program logics

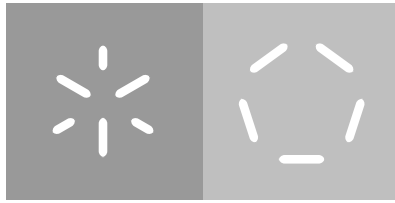
Leandro Rafael Moreira Gomes

This work was funded by the ERDF—European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation—COMPETE 2020 Programme and by National Funds through the Portuguese funding agency, FCT—Fundação para a Ciência e a Tecnologia, within projects POCI-01-0145-FEDER-016692, POCI-01-0145-FEDER-016826, POCI-01-0145-FEDER-030947 and POCI-01-0145-FEDER-029946



UIMinho | 2022

May 2022



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Leandro Rafael Moreira Gomes

Weighted computations:
semantics and program logics

Doctoral thesis
Doctoral Programme in Computer Science

Thesis supervised by
Luís Soares Barbosa
Alexandre Madeira

May 2022

DIREITOS DE AUTOR E CONDIÇÕES DE UTILIZAÇÃO DO TRABALHO POR TERCEIROS

Este é um trabalho académico que pode ser utilizado por terceiros desde que respeitadas as regras e boas práticas internacionalmente aceites, no que concerne aos direitos de autor e direitos conexos.

Assim, o presente trabalho pode ser utilizado nos termos previstos na licença abaixo indicada.

Caso o utilizador necessite de permissão para poder fazer um uso do trabalho em condições não previstas no licenciamento indicado, deverá contactar o autor, através do RepositóriUM da Universidade do Minho.



Atribuição

CC BY

<https://creativecommons.org/licenses/by/4.0/>

ACKNOWLEDGEMENTS

Estaria a mentir se dissesse que durante toda a minha vida não sonhei com outra coisa a não ser fazer um doutoramento. Eu até cheguei a dizer que não era, de todo, esse o caminho que pretendia seguir para a minha carreira. Na verdade, foi de alguma forma “por acaso” que inicei esta jornada. Mas tenho descoberto cada vez mais que, de todos os momentos que vivenciamos na nossa humilde e ínfima passagem pelo universo, desde os mais efêmeros acontecimentos até às experiências mais duradouras, é precisamente naqueles mais inesperados, naqueles que mais acontecem por acaso, e na espontaneidade e incerteza que os caracteriza, que a felicidade acaba por estar presente de uma forma mais evidente.

Por isso, queria agradecer, em primeiro lugar, aos meus orientadores Professor Luís Barbosa e Alexandre, e ao Professor Manuel Martins, que foram as primeiras pessoas que me impulsionaram a iniciar este doutoramento, e com as quais partilhei a maior parte deste percurso. Em particular agradeço aos meus orientadores pela disponibilidade, exigência, compreensão e apoio, não só na parte científica em si, mas pelo que, hoje, já representam no lado mais pessoal.

Em segundo lugar, um muito obrigado a todos os meus colegas e amigos Rita, Pedro, Jifu, Guille, Carlos, Renato, Zé, Catarina e Chong, que partilharam comigo muitas das experiências ao longo destes seis anos e que, de uma forma ou de outra, com mais ou menos “peso”, acabaram por ser um apoio importante.

Finalmente, tenho que agradecer de uma forma muito especial à minha mãe, ao meu pai e à minha irmã, porque esta jornada coincidiu, infelizmente, com alguns dos acontecimentos mais difíceis pelos quais passamos juntos. A força que tivémos para os ultrapassar, juntos, serviu igualmente de sustento que alimentou este documento que aqui apresento. Agradeço, também, como não poderia deixar de ser, aos meus gatos Gato, Sininho, Júnior, Cheeta, Cheeto e Tiziu, aos quais devo o sorriso que acompanha o meu dia-a-dia. Tenho a certeza que, se hoje sou uma pessoa mais cuidadora, a eles o devo.

O último agradecimento, e a quem mais dedico este trabalho, vai para a minha avó que, não podendo estar presente fisicamente nesta parte final da caminhada, foi graças ao seu grande apoio e carinho que consegui superar algumas pedras que encontrei nos primeiros trilhos. Eu sei que ela, de onde me estiver a ver, estará orgulhosa.

This work was funded by the ERDF—European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation—COMPETE 2020 Programme and by National Funds through the Portuguese funding agency, FCT—Fundação para a Ciência e a Tecnologia, within projects POCI-01-0145-FEDER-016692, POCI-01-0145-FEDER-016826, POCI-01-0145-FEDER-030947 and POCI-01-0145-FEDER-029946

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

COMPUTAÇÕES PESADAS: SEMÂNTICAS E LÓGICAS DE PROGRAMAS

Esta tese debruça-se sobre computações pesadas ou, por outras palavras, programas e asserções sobre estes cuja execução ou avaliação tem alguma forma de *peso* associado. Por peso queremos dizer um valor que pode representar, por exemplo, uma incerteza na execução, ou uma quantidade de recursos consumidos, como energia ou tempo. Exemplos de sistemas que contêm alguma componente com pesos variam desde comunicações entre dispositivos, processos biológicos em rede, sistemas de apoio à decisão clínica ou controlo de robots, estando cada vez mais presentes no nosso quotidiano. Neste sentido, devido à alta complexidade subjacente à introdução destes parâmetros, exige-se que a engenharia de software recorra a metodologias de desenvolvimento rigorosas para garantir a alta fiabilidade de cada produto de software. E se é verdade que o desenvolvimento, análise e verificação destes sistemas são cada vez mais assentes nessa mesma abordagem formal, as práticas correntes de programação não são ainda capazes de oferecer uma estrutura que seja, ao mesmo tempo, genérica o suficiente por forma a capturar estes paradigmas e capaz de satisfazer os requisitos específicos para cada domínio de aplicação.

Nós queremos atacar este desafio através da apresentação de uma metodologia de desenvolvimento sistemático de semânticas e lógicas para raciocinar sobre duas classes distintas de programas. Na primeira classe, a que nós chamamos de computação de *fluxo único*, cada execução é uma única transição com um peso associado. Na segunda classe, a que nós chamamos computação de *fluxo múltiplo*, cada execução pode assumir múltiplos caminhos em simultâneo, cada um com um peso possivelmente distinto. Nesta tese definimos, para cada classe de computação, uma semântica, e provamos que esta forma uma álgebra apropriada para raciocinar sobre programas dessa classe. Para esse fim, definimos operadores que interpretam as construções básicas de uma linguagem de programação imperativa: composição sequencial, condicionais e iteração. A partir daqui construímos uma lógica, incluindo o respetivo sistema axiomático, que permite verificar propriedades sobre estes programas.

Uma das virtudes desta metodologia é a sua parametricidade, que é dada por uma estrutura matemática genérica que oferece tanto um modelo de computação para representar programas como um espaço de verdade para avaliar asserções sobre eles.

Para a classe de computações de *fluxo único*, definimos também uma noção de bisimilaridade nos modelos dos lógicas geradas, provando-se invariância modal para essas lógicas.

Palavras-chave: álgebra de Kleene, computação pesada, lógica dinâmica, semânticas de programas.

WEIGHTED COMPUTATIONS: SEMANTICS AND PROGRAM LOGICS

This thesis deals with weighted computations or, more precisely, programs and assertions about them whose execution or evaluation has some form of *weight* associated. By weight we mean a value which may represent, for example, an uncertainty in the execution, or a quantity of resources consumed, such as energy or time. Examples of systems containing some component with weights range from device-to-device communications, network biological processes, clinical decision support systems or robot control, being these growingly present in our everyday life. In this sense, due to the complexity underlying the introduction of these parameters, software engineering is forced to call upon rigorous development methodologies which provides a high assurance of each software product. And if it is true that the development, analysis and verification of these systems are increasingly laid on this exactly approach, the current programming practices are not capable to offer a framework which is, at the same time, generic enough to capture such paradigms, and able to satisfy the specific requirements for each application domain.

We intend to address this challenge by presenting a methodology for the systematic development of semantics and logics to reason about two distinct classes of programs. In the first class, that we call *single-flow* computation, each execution is a single transition with an associated weight. In the second class, that we call *multi-flow* computation, each execution may assume multiple simultaneous execution paths, each one of them with a, possible distinct, weight. In this thesis we define, for each class of computation, a semantics, and we prove that it forms a suitable algebra to reason about programs of that class. For that end, we define operators which interpret the basic constructions of an imperative programming language: sequential composition, conditionals and iteration. From here we construct a logic, including the respective axiomatic system, allowing to verify properties over those programs.

One of the merits of this methodology is its parametricity, which is given by a generic mathematical structure offering both a computational model to represent programs and a truth space to evaluate assertions over them.

For *single-flow* computations, we define as well a notion of bisimilarity on the models of the generated logics, and prove the modal invariance property for those logics.

Keywords: dynamic logic, Kleene algebra, program semantics, weighted computation.

CONTENTS

1	Introduction	1
1.1	A new challenge for computational systems	1
1.2	Goal	3
1.3	Illustration	5
1.4	State of the art	6
1.5	Contributions	15
2	Background	23
2.1	Propositional dynamic logic	23
2.2	Kleene algebra and action lattice	26
2.3	Generating multi-valued propositional dynamic logics	31
2.4	Weighted sets and weighted relations	35
I	Weighted <i>single-flow</i> computations	
3	Algebras of weighted <i>single-flow</i> computations	41
3.1	Preliminaries: Kleene algebra with tests and Hoare logic	41
3.2	Generalising Kleene algebra with tests: a first approach	43
3.3	Generalising Kleene algebra with tests: an idempotent variant	50
3.4	Weighted structures and matrices as GKAT/I-GKAT	53
3.5	An illustration: a folk theorem	66
4	A weighted <i>single-flow</i> semantics	71
5	Dynamic logics for weighted <i>single-flow</i> computations	79
5.1	Generation of multi-valued equational dynamic logics	79
5.2	Back to the weighted single-flow semantics	95
5.3	An illustration	98
5.4	Bisimulation	100
II	Weighted <i>multi-flow</i> computations	
6	Algebras of weighted <i>multi-flow</i> computations	108

6.1	Preliminaries: binary multirelations	108
6.2	Introducing weighted multirelations	111
7	A weighted <i>multi-flow</i> semantics	126
8	Dynamic logics for weighted <i>multi-flow</i> computations	135
8.1	Generation of *-free multi-valued equational dynamic logics	135
8.2	Back to the weighted multi-flow semantics	140
8.3	An illustration with fuzzy Arden syntax	146
8.4	An illustration with jFuzzyLogic	150
9	Conclusions and open problems	158
	Bibliography	165

LIST OF FIGURES

Figure 1	Schematic diagram of <i>D2D</i> mobile communication.	2
Figure 2	Weighted “single-flow” and “multi-flow” computations. The values a and a_1, a_2, \dots, a_n represent weights.	4
Figure 3	Transition system correspondent to Example 1.3.1.	5
Figure 4	Action lattice as a combination of Kleene algebra and residuated lattice	9
Figure 5	Binary tree representing successive (possible biased) coin tosses.	10
Figure 6	Conditional in a fuzzy programming language, illustrated by a liquid flowing through a ‘Y-shaped’ pipe.	18
Figure 7	Illustration of $\mathbf{L}^{W \times 2^W}$	20
Figure 8	Illustration of $\mathbf{L}^{W \times L^W}$	20
Figure 9	Hoare logic rules.	42
Figure 10	Examples of KAT, GKAT and I-GKAT.	51
Figure 11	Representation of the variable <code>02_low</code>	128
Figure 12	Graph <code>02_low</code> (blue line) and <code>02_low+5</code> (red line)	129
Figure 13	Human body temperature variation in a 24h period	131
Figure 14	Blood pressure variation in a 24 hour period	132
Figure 15	Graph distance between crane head and target position	152
Figure 16	Graph angle of the container to the crane head	152

LIST OF TABLES

Table 1	Taxonomy of related work of this thesis	14
Table 3	Syntax of \mathcal{F}_1	20
Table 2	Taxonomy of related work and the frameworks introduced in this thesis	21
Table 4	Syntax of \mathcal{F}_2	21
Table 5	Satisfiability	161
Table 6	Global satisfiability	161
Table 7	Taxonomy of related work, frameworks introduced in this thesis and additional ones we left for the future	164

INTRODUCTION

1.1 A new challenge for computational systems

It is consensual that a rigorous design discipline is crucial to boost productivity and enforce correctness in software production. Thus, the development of formal techniques and tools for both program specification and verification go hand in hand. For example, the use of logics to perform the formal verification of the following simple program

Example 1.1.1.

```
while  $y \neq 0$  do  
begin  $z := x \bmod y$ ;  
 $x := y$ ;  
 $y := z$ ;  
end
```

which computes the greatest common divisor of two integers, lies on the prior definition of some sort of specification of its behaviour. The idea is to provide obligations that need to hold after the execution of the program, assuming that some conditions are verified beforehand. Dynamic logics [Pra91], Hoare logic [Hoa69] and Kleene algebra [Kle56] are among the first and well established formal approaches to verify classic imperative programs, as the one of Example 1.1.1.

Often, however, the behaviour of computations is weighted by some factor e.g. a probability, a cost, a measure of uncertainty or energy consumed, or even execution time, so that reasoning about it requires taking such weights seriously. In particular, reasoning with uncertainty is essential for a wide variety of application domains, such as recommender systems [CL14], image detection algorithms [KTMK15], robotics [CAf13, CSLM19], pharmaceutical applications [WLF⁺20] or support for medical diagnosis [VMA10]. An example of the latter is the design of a system that helps with clinical decision, e.g. a program which decides whether a patient needs some drug depending on its fever condition. Defining the threshold of body temperature that is worth of attention represents the type of reasoning that cannot be simply set in Boolean terms, i.e. true or false. Otherwise, the system would risk ignoring, for example, a patient with a body temperature very close to fever condition, but still not above the threshold.

Another form of weight relevant in the software development process is the measure of computational costs or energy consumed along the execution of some program. These are typical concerns of *green*

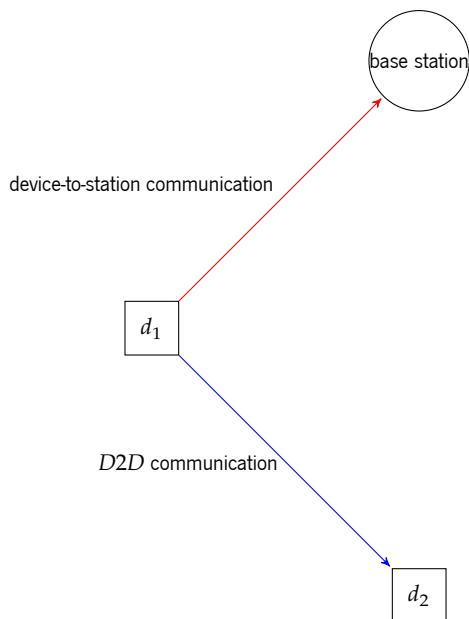


Figure 1: Schematic diagram of *D2D* mobile communication.

computing, whose main goal is to maximize energy efficiency and achieve more sustainable software and hardware development practices [Kur08]. A rich research agenda flows from this philosophy, aligned with the crescent worry for the global climate changes, with major focus on reducing energy consumption of large data centers [BS09], as well as on the development of energy-aware methodologies for small devices, interconnected through e.g. 5G mobile networks [WLCW20, HCHH19].

For example, new developments in mobile transmission techniques, regarding *device-to-device* (*D2D*) interaction lead to new possibilities for communication between devices, with positive implications in the efficiency of the network. In a concrete situation where two handheld devices that are in physical proximity use high data-demand services, like multiplayer gaming or video sharing, *D2D* communication between those devices could relieve great network traffic from a base station, thus increasing its energy efficiency.

Let us consider the following scenario, depicted in Figure 1: two devices, d_1 and d_2 , are in physical proximity and intend to join a multiplayer gaming session, where they need to be in constant communication with each other to transmit data. Suppose that device d_1 initiates the communication and thus should decide, in real time, whether the data transmission is relayed by the base station or directly with device d_2 , supported by *D2D* technology. In such a situation, the measure of resources consumed is of high importance for making the best decision to enhance the user experience, for example, by reducing system latency and the base station traffic. One may formalise such a problem, in very generic terms, as the computation of the communication path which minimizes the usage of such resources.

The nature of these systems, which rely on some (non trivial) form of weighted computation, entails the need not only for programming languages which can describe their behaviour, but also for semantic structures and logics on which to base their rigorous design, verification and refinement.

The focus of this thesis is precisely systems whose behaviour is explained by some form of weighted transitions. The computing paradigms embedded in such systems emerged essentially as formalisations of knowledge for expressing concepts that cannot be evaluated in simple Boolean terms. While in the

“classical world”, to which programs as the one in Example 1.1.1 belong, Kleene algebras and dynamic logics provide the underlying formal setting for correctness in software production, a generic framework tuned to weighted computing paradigms mentioned above is lacking. To deal with the complex, often unpredictable, behaviours typical of such paradigms, it is relevant to establish a generic formal setting, i.e. i) algebras to model classes of computations where weights are a natural ingredient; and ii) logics, to provide a finite set of rules for reasoning about those computations.

How to come up with such a framework is the overall aim of the thesis.

1.2 Goal

The basic challenge in this PhD is to provide a rigorous method for the design and verification of two classes of imperative programs, on top of weighted computations, which model the two paradigms depicted in Figure 2.

The first class consists of programs interpreted as weighted “*single-flow*” computations, i.e. each program is a *single* transition between two states in a transition system, with an associated weight. Those weights, depending on the associated semantic domain, may represent the uncertainty of the execution, a cost or a quantity of resources consumed, or even the execution time.

The second class concerns programs interpreted as a transition from a state to a weighted set of states, modelling a sort of parallel run typical of conditionals in programming languages designed for fuzzy control systems. In such statements, the branches are executed simultaneously with (possibly) different weights, given by the evaluation of each condition. Such a behaviour is precisely what we mean by weighted “*multi-flow*” computation.

Figure 2 schematizes the general idea which formally represents “weighted single-flow” and “weighted multi-flow” computations, along with a comparison with the corresponding classical case.

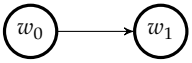
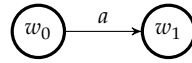
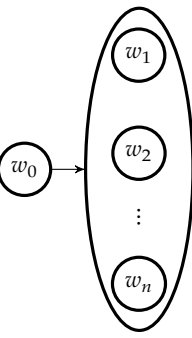
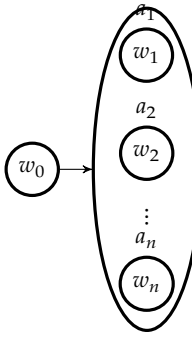
	Classic	Weighted
“Single-flow” (Relational)	$2^{W \times W}$ 	$M^{W \times W}$ 
“Multi-flow” (Multirelational)	$2^{W \times 2^W}$ 	$2^{W \times M^W}$ 

Figure 2: Weighted “single-flow” and “multi-flow” computations. The values a and a_1, a_2, \dots, a_n represent weights.

The classes of programs described above are represented, formally, by the second column of the table. The main idea is to take as a starting point in the classical notion of program as a binary (multi) relation, generalising then its semantic domain to capture weighted computations. By embedding a proper structure of *weights* into a binary relation, we obtain the formal model of *weighted “single-flow” computations*, represented in the upper right cell of the table. By, instead, treating a program as a transition from a state to a *weighted* set of states, we obtain the formal model of *weighted “multi-flow” computations*, represented in the lower right cell of the table.

For each kind of computation described above the goal of this thesis is to characterise i) the mathematical structure that acts as a parameter in the development of ii) a formal semantics for programs interpreted as weighted “single-flow” and weighted “multi-flow” computations, and iii) a systematic method of generating dynamic logics, for the verification and analysis of programs of these classes.

The development of such a framework will be set up on generalisations of the well established notions of Kleene algebra and dynamic logic, on the algebraic and logical side, respectively. This approach relies on the definition of a structure of weights for each kind of computation acting, in both cases, simultaneously as a computational model to interpret programs, and as a truth space to evaluate assertions about those programs. How to achieve this goal constitutes the challenge addressed in this thesis, taking the following research questions as a guideline.

- Which are the suitable semantic structures and logics to reason about weighted “single-flow” and “multi-flow” computations?
- Which properties can be verified in these settings?

1.3 Illustration

Let us now provide a simple example to illustrate each class of programs described above. The first class is composed by programs interpreted by *weighted “single-flow”* computations. Consider, for example, the following imperative program

Example 1.3.1.

```
double x := 2; double x := x + y;
if (x ≤ 3)
x := x + 1;
else
y := y × 2;
```

which corresponds to the transition system depicted in Figure 3. Given a set of variables X and a mathe-

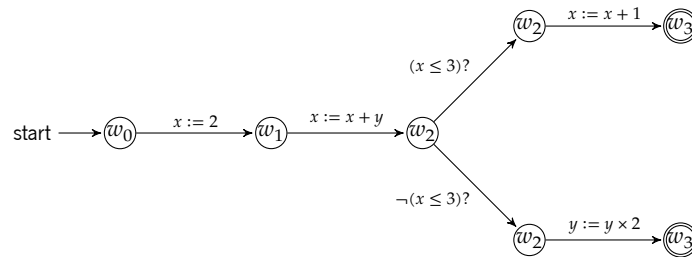


Figure 3: Transition system correspondent to Example 1.3.1.

tical structure \mathbf{A} to model the space of weights, states are functions $w_i : X \rightarrow \mathbf{A}^{\mathbb{R}}, i \in \mathbb{N}$, assigning a value in a proper weighted truth space to a variable x in a given data domain, in this case, the real numbers \mathbb{R} . For a mere illustrative example, we may consider, as the domain of weights, the energy consumed by a system. Thus, $w_0(x)(1)$ may represent, for instance, the energy consumed when assigning the real value 1 to the variable x in the initial state w_0 .

The simplest case of a programming construct is an assignment $x := 2$, whose weight in this context is a measure of the energy consumed by performing the assignment. Other more complex program constructs can also be defined, such as sequential composition and conditional. An example of the former is the program $x := 2; x := x + y$, which accumulates the total energy used by performing sequentially the atomic programs $x := 2$ and $x := x + y$. An instance of the latter is the conditional statement $(x \leq 3)?; (x := x + 1) + (\neg(x \leq 3))?; (y := y \times 2)$, interpreted as the minimum energy consumed by one of the assignments $x := x + 1$ and $y := y \times 2$, guarded by the predicate $x \leq 3$.

The other class of programs which are the object of study of this thesis consists of ones interpreted as *weighted “multi-flow”* computations, typically designed for fuzzy control systems [Ost82], with applications to e.g. medical diagnosis [VMA10], similar to the example described in Section 1.1, or robotics [CAf13]. The *fuzzy Arden syntax (FAS)* [VMA10] is an example of a fuzzy programming language used in the design

of fuzzy control systems for clinical decision support. let us consider the simple FAS program represented below.

Example 1.3.2.

```
if (Temperature is in Fever_condition)
then medicine:=5 else medicine:=0
```

In simple terms, the program makes an adjustment of the dose of medicine to be administrated to a patient depending on his or her temperature. The latter, formally, is a function assigning, to each (crisp) value of temperature, a real value (e.g. in the range $[0, 1]$) to record how close such temperature is from a “fever condition”. In a scenario where the predicate `Temperature is in Fever_condition` and its negation assume a value greater than 0, let us say, 0.4 and 0.6, respectively, we obtain a somehow ambiguous evaluation, and therefore the program executes both the `then` and the `else` blocks, weighted by the value of each condition. In practice, this results in a multiplication of the variable `medicine`. Intuitively, the values 0.4 and 0.6 mean that `Temperature` has probably not reached the limit of a fever condition but is close to it.

In other words, the intended semantics of a “conditional” in FAS does not reduce to a non deterministic or even a probabilistic choice [MRS13]. Instead, that of it corresponds to a sort of parallel run in which the branches are executed with a (possible) different weight, given by the evaluation of each condition. This also differs, of course, from the usual parallel composition which models a crisp notion of two programs running in parallel [FS15, HMSW11, Pel87], or two actions being processed at the same time [Pri10].

1.4 State of the art

It is consensual that a rigorous design discipline is crucial to boost productivity and enforce correctness in software production. A vast myriad of approaches, based on diverse forms of Hoare logics, Kleene algebras and dynamic logics, or combinations thereof, have been proposed over the years for this purpose. Writing a correct program is achieved by first specifying how the program is supposed to behave, through a *correctness specification*. For the program of Example 1.1.1 such a specification may be the following assertion:

If the input values of x and y are positive integers c and d , respectively, then

- the output value of x is the *gcd* of c and d , and
- the program halts.

The specification consists of a *precondition* ρ_1 and a *postcondition* ρ_2 , which defines properties of the input and output states of the computation, respectively. If a program, starting on a state satisfying

ρ_1 , halts in a state satisfying ρ_2 , it is said that the program is *correct* with respect to the input/output specification ρ_1, ρ_2 . For example, for program of Example 1.1.1, we may consider the precondition

$$(x \geq 0 \wedge y > 0) \vee (x > 0 \wedge y \geq 0)$$

and the post condition $gcd(x, y)$. The program is correct with respect to such specification.

Hoare logic was one of the first formal systems proposed for verification of programs with respect to partial correctness. Introduced in 1969 by Hoare [Hoa69], its wide influence made it a cornerstone in the field. In Hoare logic, the specification of a program is written as *partial correctness assertions* of the form $\{\rho_1\}\pi\{\rho_2\}$, also called *Hoare triples*. Symbols ρ_1 and ρ_2 stand for the precondition and the postcondition, respectively, and π is the program statement. The intuitive meaning of a Hoare triple is the following: a program is correct if the satisfaction of precondition ρ_1 implies that for every execution of π , if it halts, the postcondition ρ_2 is satisfied. The correctness of a program specified as a Hoare triple is verified in Hoare Logic, through a set of inference rules [Hoa69, Flo93]. A fragment of Hoare logic is *Propositional Hoare logic (PHL)* [Koz00], in which Hoare triples are reduced to syntactic assertions.

The *predicate transformer semantics* was introduced by E. Dijkstra [Dij76] to interpret each program statement in an imperative programming language as a function between two predicates, called *predicate transformer*. Different semantics for such a function can be defined: *weakest preconditions*, which give the less restrictive condition to assure that a program satisfies some predicate after running; and *strongest postconditions*, giving the strongest condition after the execution of π under a given precondition. Alternatively the predicate transformer semantics can be viewed a reformulation of Hoare logic, transforming the challenge of verifying the validity of a Hoare triple into proving a first-order logic formula. The relation to partial correctness in Hoare logic is given in the form of *weakest liberal precondition (wlp)*, a function yielding the weakest condition under which a program $\pi \in W \times W$ either does not terminate or terminates in a state satisfying $\rho \subseteq W$, formally

$$wlp(\pi, \rho) = \{w \in W : \pi(w) \subseteq \rho\}$$

Other formalisation presents $wlp(\pi, \rho)$ as the largest subset $V \subseteq W$ such that $\pi(V) \subseteq \rho$. Another predicate transformer is *weakest precondition (wp)*, differing from wlp by assuring termination of π .

In order to use Hoare logic for the verification of a program, one needs first to formalise the program behaviour. For instance, the program of Example 1.1.1 can be seen as an input/output relation of pairs (a, b) of states which relates the input with the output of the program. The mathematical structure that can formalise such an interpretation is that of a *binary relation* which, together with the operators of relational composition and set union, provides the necessary ingredients to interpret the notion of sequential composition, iteration and nondeterminism.

Binary relations can be structured as a *Kleene algebra*. Emerging as a generalisation of regular expressions [Kle56], Kleene algebra plays however a most relevant role in semantics of programs [Koz82, Pra88]. With numerous extensions and applications (e.g. [QWWG08, FD07, AFG⁺14, Koz14, FKM⁺16]) Kleene algebra is a standard structure to reason about programs, providing a finite axiomatisation precisely to rea-

son about sequential composition, iteration and nondeterminism. Nevertheless the abstraction of some programming constructs, like **if-then-else** and **while** statements, require not only the algebraic formalisation provided by Kleene algebra, but also a rigorous way to include tests.

How to include those assertions into a model of computations is a challenge that gave rise, over time, to many distinct approaches and results. One such concretisation is *dynamic logic* [Pra76], a very powerful framework to reason about programs in a precise way, allowing to interpret both its object (i.e. the very notion of a program, modelled by a Kleene algebra) and assertions over it. These two components can be, respectively, *abstract programs* and *propositions*, in *Propositional Dynamic Logic (PDL)* [FL77], or lifted up to *assignment statements* and *formulas* built over variables and terms, in *First-Order Dynamic Logic* [HKT00].

The syntax of PDL simply abstracts predicates and programs into propositional symbols. It represents a combination of propositional logic and the algebra of regular expressions. First-order dynamic logic, on the other hand, bases its syntax on *first-order logic*. It consists of expressions built over a set of *variables*, *terms*, and *function* and *predicate* symbols. On top of this vocabulary, sets of programs and formulas are defined. The most simple notion of a program, i.e. an atomic program, consists of an assignment of the value of a term t to a variable x , denoted as $x := t$. A formula is built over terms t_1, \dots, t_n and predicate symbols p , denoted as $p(t_1, \dots, t_n)$.

As a consequence of such differences in the expressibility, while PDL considers states as mere abstract points, in first-order dynamic logic they consist on valuations of variables in a given data domain. Moreover, compound programming constructs, like sequentiality, iteration and nondeterminism are inductively built from atomic programs, with syntax built over a previously defined signature (propositional or first order).

Both PDL and first-order dynamic logic need nevertheless more expressibility to establish the interaction between programs and assertions. On the assertions side, and based on modal logic [BvBW06], formulas of dynamic logic contain also the *modalities* $\langle \rangle$ (possibility) and $[\]$ (necessity) to reason about the effects of program executions in the states of computation. On the programs side, a proper notion of *test* is required to reason about programming constructs depending on assertions, namely **if ρ then π_1 else π_2** and **while ρ do π** . A test is a proposition/formula accompanied by the operator $?$, which transforms propositions/formulas into programs.

Dynamic logic is also expressive enough to subsume the syntax and the inference rules of Hoare logic, reducing its deductive apparatus to the validity of a dynamic logic formula. More concretely, the validity of a Hoare triple $\{\rho_1\}\pi\{\rho_2\}$ corresponds to the validity of the dynamic logic formula $\rho_1 \rightarrow [\pi]\rho_2$. As an example, to verify the correctness of the program of Example 1.1.1, one should replace ρ_1 by $(x \geq 0 \wedge y > 0) \vee (x > 0 \wedge y \geq 0)$ and the ρ_2 by $gcd(x, y)$ and π by the program, and reason about the validity of the resulting formula, using an appropriate semantics.

The automation of program correctness using dynamic logic becomes, in a practical sense, essential to provide a complete resource window for the use of this logic in program verification. The KeY project is an example of such a tool. In particular, it is a deductive verification framework specialised in proving correctness of Java programs [BKW16], supporting both interactive and fully automated proofs. Some extensions to KeY have been introduced to support also the correctness of C programs and hybrid systems.

Alternatively to PDL, the propositional abstraction of programming constructs, such as conditional and while loops can be formalised in Kleene algebra with tests (KAT) [Koz00], i.e. a Kleene algebra with an embedded Boolean subalgebra to model assertions. Like PDL, KAT is able to subsume the syntax and the inference rules of PHL, reducing its deductive apparatus to (quasi) equational reasoning. Therefore, the validity of a Hoare triple $\{\rho_1\}\pi\{\rho_2\}$ corresponds to the validity of the PDL formula $\rho_1 \rightarrow [\pi]\rho_2$ and in its turn to the equality $\rho_1; \pi = \rho_1; \pi; \rho_2$. The axiomatisation of KAT provides the necessary rules to manipulate programs as equations and verify their correctness. One particular application of KAT in program manipulation is equational proofs of program equivalences, which is presented in reference.

An extension to Kleene algebra was introduced in reference [Koz94b] to overcome the limitations of action algebras [Pra91] in their applicability to certain automata constructions and algorithm analysis. The structure, called *action lattice*, is a subclass of action algebras. As depicted in Figure 4, it combines

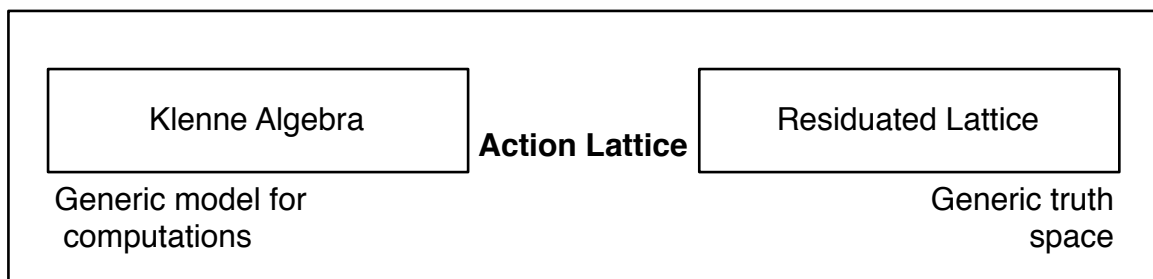


Figure 4: Action lattice as a combination of Kleene algebra and residuated lattice

Kleene algebras with residuated lattices into a single structure: as a Kleene algebra, it represents a model for computations; as a residuated lattice, it satisfies the properties to provide a multi-valued truth space to evaluate assertions about computations.

A distinct approach to marry programs and logics to reason about them aims at a cooperation between KAT and PDL [DMS06]. Concretely, this research line introduces *Kleene algebra with domain (KAD)*, an extension of Kleene algebra by a map, called the *domain operator*, linking the worlds of programs and tests. This structure acts in two directions: it allows to embed propositions into actions, as in KAT, and to map programs to propositions, as in PDL. The domain operator behaves as the modal diamond operator of dynamic logic. KAD provides an algebraic framework for reasoning about programs with this modality. For example, the PDL formula $\langle \pi \rangle \rho$ is semantically equivalent to the expression $d(\pi\rho)$ in KAD: if represented relationally, both are the preimage of the states satisfying ρ under program π . One particular application of this structure is to prove the algebraic soundness and completeness of Hoare logic deductive system [MS04].

All these mathematical formalisations are suitable to reason about classic systems typically modelled by labelled transition systems (LTS), where assertions over programs are stated in Boolean logic. Over time, the interpretation of a program evolved in unexpected ways, leading to the development of diverse algebras and logics tailored to more complex programming paradigms, such as illustrated in Section 1.3. Such structures include, naturally, an entire family of dynamic logics (e.g. [FH84, Koz85, BS06, Pla10,

MNM16, Sed20]), and extensions and generalisations to Kleene algebra (e.g. [MCM06, QWWG08, QWG08, AFKW14, AFG⁺14, FKM⁺16]).

Possibly the most common ingredient of such complexity arises in the form of uncertainty, interpreted in most scenarios as a probability. Probabilistic programs extend classic imperative programs with randomization, i.e. assignment of a random value to a variable according to a probability distribution. This notion can be introduced in two ways: as an *assignment* $x := ?$, and as a *probabilistic choice* $\pi_1 +_\alpha \pi_2$ between programs π_1 and π_2 . The former assigns to x a set of values with a uniform probability distribution. The latter is interpreted as a probability distribution over sets of possible terminating states. In other words π_1 is obtained with probability α and π_2 with probability $1 - \alpha$. An example of such a program is

$$(x := \text{Heads}) +_\alpha (x := \text{Tails}) \tag{1}$$

representing a (possibly biased) coin toss. Each run of this program can be intuitively interpreted as a “yes-no” question, which can be modelled by a Bernoulli distribution of a random variable. The answer to this question (the output of the program) is a single terminating state obtained from a probabilistic distribution, assuming the value Heads with probability α and Tails with probability $1 - \alpha$. In this scenario, even with the output of a program being a probability distribution, each run of the program entails either $x := \text{Heads}$ or $x := \text{Tails}$. Successive runs of this program can be modelled by the binary tree of Figure 5 (see reference [MMS96] for examples and discussion).

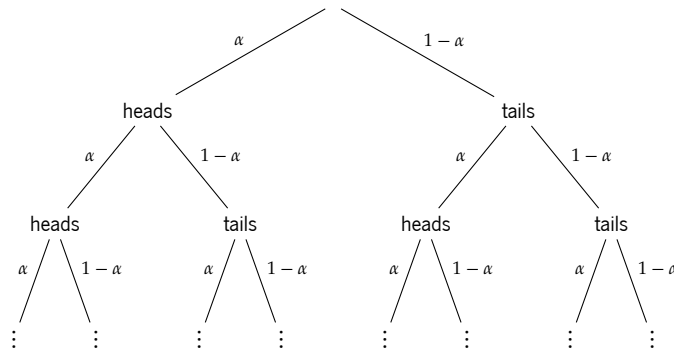


Figure 5: Binary tree representing successive (possible biased) coin tosses.

Numerous mathematical abstractions of probabilistic programs have been studied over the last few decades, namely program algebras [MCM06, QWWG08], semantics [Koz81, HSM97] and logics [Koz85, dHdV02, QWG08]. Two equivalent semantics for an abstract probabilistic programming language were given in reference [Koz81]. One interprets programs as partial measurable functions over measurable spaces, while the other defines programs as linear operators on Banach spaces of measures. Shortly after, the same author introduced a probabilistic version of PDL, *probabilistic propositional dynamic logic (PPDL)*, supported by the previously introduced semantics, to reason about probabilistic programs [Koz85]. The notion of satisfiability is represented as the expected value of a given proposition f in a state μ , calculated as the integral $\int f d\mu$. Reference [FH84] adapts Kozen’s semantics of PPDL to formulas involving probabilistic programs. In another context, the work of [MMS96] extends predicate transformers to reason about

probabilistic programs with respect to pre and postconditions. We also put a remark on a notion that slightly relaxes probabilistic choice [CW08], defined *sub-probabilistic choice*, i.e.

$$(\alpha)\pi_1 + (\beta)\pi_2 \tag{2}$$

where $\alpha + \beta \leq 1$. The expression (2) means that program π_1 is executed with probability α and program π_2 with probability β , without imposing $\alpha + \beta = 1$. Two distinct semantics are provided. One interprets programs as sub-probability distributions, as functions $\llbracket \pi \rrbracket : W \rightarrow \mathbf{D}(W)$, where

$\mathbf{D}(W) := \{\delta : W \rightarrow [0, 1] \mid \sum_{w \in W} \delta(w) \leq 1\}$ is the set of sub-probability distributions δ over W . The other interprets programs as bounded expectation transformers in terms of *wp*-semantics, i.e. functions $\alpha : W \rightarrow \mathbb{R}^+$ bounded by a value $M \in \mathbb{R}^+$ such that $\alpha(w) \leq M$. Soundness and completeness for total correctness is also proved for such class of programs, in terms of triples $\alpha\{\pi\}\beta$. The validity of such a triple is stated in terms of an *expected value* $l \in \mathbb{R}^+$: if the precondition α is satisfied in w with expected value l , then program π terminates in w and its output satisfies the postcondition β with l too. Thus the meaning of a weakest precondition $wp(P, \beta)$ is the weakest probabilistic predicate α that makes the triple $\alpha\{\pi\}\beta$ valid.

A Hoare-like logic was also proposed to account for partial correctness of probabilistic programs with a first-order syntax [dHdV02, RZ15]. These references use probabilistic predicates over random variables. An example of the probabilistic predicate is $P(x = 1) = \frac{1}{2}$ states that $x = 1$ with probability $\frac{1}{2}$. It is relevant to note, nevertheless, that only one course of execution is taken. A propositional abstraction of a probabilistic program is given in reference [QWWG08], with the introduction of *probabilistic Kleene algebra with tests (PKAT)*, an extension of KAT with an operator for probabilistic choice. The structure captures precisely programs like (1). An operational semantics and a probabilistic bisimulation equivalence relation for PKAT are also given in the paper. Although the notion of a probabilistic bisimulation was first mentioned in [BM89], the standard definition is the one used in [LS91]. It is defined as an equivalence relation on a set of states W , stating that two states $w_1, w_2 \in W$ are bisimilar if and only if w_1 relates with state w through an action a with a probability α implies w_2 relating with w with the same probability α and vice versa, formally $w_1 \xrightarrow{\alpha} w \Leftrightarrow w_2 \xrightarrow{\alpha} w$.

A survey on distinct notions of probabilistic bisimulation for variants of probabilistic transition systems can be found in [SdV04]. The authors of [QWWG08] discuss, in another paper, an encoding of a while-free fragment of Hoare logic in PKAT [QWG08].

A next, although not much obvious, step was to develop rigorous mathematical structures and logics to reason about programs containing both nondeterministic and probabilistic choices. The first type of nondeterminism may have two variants: angelic and demonic. The former happens when the choice is made by an ‘angel’, which corresponds to the best possible situation from a set of available scenarios. The latter corresponds to a choice made by a ‘demon’, occurring when the outcome is the worst possible scenario. Since the decision is made by an external agent, e.g. a computer during a program run, we never know which option will be chosen. Moreover, each initial state may have more than one terminating

state. To illustrate, below we present an example containing demonic nondeterministic and probabilistic choices [MM01a].

Consider the flip of a coin with the state space $s : \{Heads, Tails\}$. The program

$$s := Heads \sqcap [s := Heads +_{\frac{2}{3}} s := Tails] \quad (3)$$

where \sqcap represents a demonic nondeterministic choice and $+_{\frac{2}{3}}$ acts as the probabilistic choice referred in (1), assigns a probability of at least $\frac{2}{3}$ to $s := Heads$. Since the choice \sqcap is demonic, we can only be sure that $Heads$ is chosen $\frac{2}{3}$ of the time.

The research line on programs containing both nondeterministic and probabilistic choices is pursued by references [MM01a, MM01b]. The first shows that nondeterministic, probabilistic, angelic and demonic choices can coexist in a system, and presents a predicate transformer semantics for programs describing those systems. The latter introduces axiomatic and operational frameworks for partial and total correctness of nondeterministic probabilistic programs.

References [MNMB15, MNM16] follow, however, another direction, by adopting a more generic interpretation for weight. The scope goes beyond the notion of weight as a probability: it can be a claim about the vagueness of an execution, resources consumed or even execution time. Abstractly, such space of weights is given by an action lattice, that acts on two dimensions: it is both a Kleene algebra to model computations and a residuated lattice to provide a (multi-valued) truth space to evaluate assertions over those computations. With a semantics established over this structure, the paper proposes a parametric, systematic method of generating multi-valued dynamic logics. This exercise is inspired by [Fit91, Fit92a], which investigates many-valued modal logics that allow both formulas and accessibility relations between worlds to be many-valued. Another approach on this direction is reported in [Sed20], in which both the semantics of transitions and the truth space of propositions are values in a mathematical structure, in this case a FL-algebra.

Following previous research on KAD, some generalisations emerged also to capture weighted computations. Reference [DS11] proposes a new axiomatisation for domain and codomain operators, leading to a relaxation of KAD, of which Heyting algebras are special cases. Later, reference [DM14] explores a weaker version of this structure, by investigating variations of domain and codomain operators to provide applications to fuzzy relations and matrices. The approach considers an idempotent left semiring (lL-semiring) as the base algebraic structure, i.e. a weaker idempotent semiring in which left distributivity of multiplication over addition and right annihilation of zero are not axioms. The structure was introduced in [Mö07] with the motivation to reason about finite and infinite streams, by allowing infinite computations which are left annihilators of sequential composition. This aims to study, for example, operational semantics of the guarded command language [Luk93, Luk94] or computation calculus [Dij98, Dij00].

The growing complexity of programs was also accompanied by the introduction of concurrency in their executions, a component ubiquitous to many current systems. Therefore the verification and analysis of concurrent programs started to be tuned to such computational paradigm, leading to the introduction of new concepts, or to the adaptation of existing ones.

While originally introduced as an alternative to predicate transformer semantics, the concept of *binary multirelation* [Rew03] was established as a formalisation of parallelism. Formally, it is a subset of $A \times P(A)$, for a set A , representing intuitively a parallel execution from a state in A to a set of states over A . Multirelations appear also in the formalisation of languages with commands having both angelic and demonic types of nondeterminism [MCR07].

Another use for multirelations goes back to the semantics of Peleg's *concurrent propositional dynamic logic (CPDL)* [Pel87], to interpret a program as a relation between an initial state and a set of terminating states. In such a setting the interpretations of sequential composition and the modalities of the logic are distinct from the corresponding cases in classic PDL, since they operate over a set of states instead of a single state. The sequential composition of two multirelations $R, S \subseteq A \times P(A)$, for a set A , is introduced as

$$R \circ S = \{(w, U) \mid \exists V \cdot (w, V) \in R \wedge \exists_{F:V \rightarrow A} \cdot (\forall_{v \in V} \cdot (v, F(v)) \in S) \wedge U = \bigcup F(V)\}$$

and the diamond operator $\langle \rangle$ of dynamic logic is defined as

$$w \models \langle \pi \rangle \rho = \{w \mid \exists U \cdot (w, U) \wedge \forall_{u \in U} u \models \rho\}$$

A pair (w, U) is in $R \circ S$ if w is related by R with some intermediate set V and each element in V is related by S to a set $F(V)$ such that $U = \bigcup F(V)$. Additionally, the formula $\langle \pi \rangle \rho$ holds in state w if there is some state U related to w and every element in U satisfies ρ . Beyond Peleg's definition of sequential composition of binary multirelations, there are two other variants of the operator, whose introduction is motivated by other contexts: the Kleisli composition and the Parikh composition. References [FS16, FKST17] provide a detailed study of their main properties. The research about KAD also gave rise to an extension of such structure with a parallel operator [FS15], where programs are, as in CPDL, modeled as binary multirelations.

Among the numerous variants of KAT, *concurrent Kleene algebra (CKA)* [HMSW11] and its extension *concurrent Kleene algebra with tests (CKAT)* [JM16] were introduced with the specific motivation of formalising concurrency. Following the solidity of KAT in reasoning about programs, these structures present a finite axiomatisation to reason about concurrent programs. They present two composition operators, representing, in program semantics, sequential and parallel compositions, linked by the *weak exchange law* $(a * b); (c * d) \leq (a; c) * (b; d)$. The semantics of CKA is given in terms of a trace model of programs. CKAT is based on a combination of two classes of automata: one is nondeterministic guarded automata, to model tests, and the other is branching automata, to model concurrency. Such an interpretation allows to process *guarded series-parallel strings*, a language model for parallel runs of a program.

An extension to CKA, combining probability with parallelism, is documented in [MRS13], named *probabilistic concurrent Kleene algebra*. The uncertainty is presented in the form of a probability, and handled through the subdistributivity law of probabilistic Kleene algebra [MM05], i.e. $p; q + p; r \leq p; (q + r)$ and the explicit probabilistic choice operator $+_{\alpha}$. The parallelism is handled by the parallel composition

operator and weakening of concurrent Kleene algebra, i.e. $p||q + p||r \leq p||q + r$. It is then proved that the set of axioms presented for such an algebra is sound with respect to a probabilistic automata model.

A relational model for probabilistic programs was introduced in [Tsu12], based on the concept of *probabilistic multirelation*. The paper takes, however, a more generic algebra to evaluate truth degrees, and agnostic with respect to some properties, such as convexity.

Summing up

The algebras and logics that we just finished describing are organised in Table 1, together with their main syntactic and semantic features. The aim of this representation is to help to distinguish, at a high level, the more classic algebras and logics from the more unconventional ones. Each row represents an entry for one

	Prop	Eq	Rel	MRel	B	W
Hoare Logic (HL) [Hoa69]		X	X		X	
Propositional HL (PHL) [Koz00]	X		X		X	
Predicate transformer (wlp) [Dij76]	X	X	X		X	
Propositional Dynamic Logic (PDL) [FL77]	X		X		X	
First-order DL [HKT00]		X	X		X	
Kleene Algebra with Tests (KAT) [Koz97]	X		X		X	
Kleene Algebra with Domain (KAD) [DMS06]	X		X		X	
Probabilistic HL [dHdV02]		X	X			X
Probabilistic wlp [MMS96]		X	X			X
Probabilistic PDL [Koz85]	X		X			X
Probabilistic KAT [QWWG08]	X		X			X
“Weighted” PDL ($\mathcal{GDL}(\mathbf{A})$) [MNM16]	X		X			X
“Weighted” KAD [DS11]	X		X			X
Algebra of binary multirelations [Rew03]	X			X	X	
Concurrent PDL (CPDL) [Pel87]	X			X	X	
Concurrent KAT (CKAT) [JM16]	X		X		X	
Concurrent Dynamic Algebra [FS15]	X		X		X	
Probabilistic CKA [MRS13]	X		X			X
Algebra of probabilistic multirelations [Tsu12]						

Table 1: Taxonomy of related work of this thesis

algebraic structure or logic well established in the literature. The different features that are discussed in this thesis are organised in pairs of columns, each pair illustrating a specific dichotomy which distinguishes the respective algebra or logic in each row. The first pair differentiates between the propositional and equational variants of a given algebra or logic. The second dichotomy divides the semantics into binary relations and binary multirelations. Third, the table helps to distinguish which semantic domain, Boolean or weighted, is captured by each framework. We organise also the different frameworks in different groups, according to their semantic features, in the following order: Boolean “single-flow”, probabilistic/weighted

“single-flow”, Boolean “multi-flow” and weighted “multi-flow”. Such a division is represented by the double horizontal lines in the table.

1.5 Contributions

As explained in Subsection 1.2, the main contribution of this thesis is the characterisation of proper mathematical structures which will provide the basis for defining relational semantics and generate dynamic logics for the modelling, analysis and verification of imperative programs with weighted “single-flow” or “multi-flow” computations.

We resort again to Figure 2 to organise the contributions of this thesis. The first column represents classic binary relations and binary multirelations, and the second column lists their generalisations, formally defined as: *weighted relations*, based on fuzzy relations introduced in [Zad65], which generalise binary relations and will be used to interpret imperative programs as weighted “single-flow” computations; *weighted binary multirelations*, which generalise binary multirelations, capture the weighted parallelism of conditionals in languages such as FAS. Our thesis is that these two mathematical concepts give precise interpretations of the programs illustrated, respectively, by examples 1.3.1 and 1.3.2.

Hence, these two structures will serve as the main ingredient for introducing semantic structures (algebraic and relational) and logics, in order to offer a framework for rigorous modelling and analysis of weighted “single-flow” and weighted “multi-flow” kinds of computations.

Part 1: weighted “single-flow” computation

Part 1 focusses on weighted “single-flow” computation. First, the algebra which acts as the computational model is obtained by investigating possible generalisations of Kleene algebra with tests, by relaxing its Boolean substructure. As a result, we obtain two structures to model computations living in multi-valued, non-probabilistic, truth spaces. Considering instances of these structures as parameters, we develop a formal semantics to model the behaviour of imperative programs as weighted “single-flow” computations, and a family of dynamic logics for their verification. The main contributions are the following.

1. (Chapter 3) *Graded Kleene algebra with tests (GKAT)* (Definition 3.2.1) and *idempotent Graded Kleene algebra with tests (I-GKAT)* (Definition 3.3.1) as generalisations of KAT, with several instantiations where computations are interpreted in a graded scenario: weighted sets (WSET) (Definition 3.4.1), weighted relations (WREL) (Definition 3.4.2) and weighted languages (WLANG) (Definition 3.4.3). Moreover, an encoding of PHL for programs and assertions interpreted in this context in both GKAT and I-GKAT. Results are stated in theorems 3.2.1 and 3.3.1. This chapter finalises with an instantiations, which revisits a classic result on denesting two while loops (Theorem 3.5.1).
2. (Chapter 4) The *semantics* of this class of programs, which consists of an interpretation for program variables, functional and predicate terms, and programs built over an equational signature. In

particular, programs are interpreted as *weighted binary relations* (Definition 2.4.2), i.e. functions which attribute a value in a *complete action lattice* \mathbf{A} to each pair of relating states of a computation. This models modelling different notions of weighted “single-flow” computations: vagueness degree associated to the effectiveness of a particular computation, a measure of the resources consumed in it, or even the associated cost or execution time.

3. (Chapter 5) A method for generating a family of *weighted equational dynamic logics*, that we call $\Gamma(\mathbf{A})$, with semantics presented in Chapter 4 and the satisfaction relation over a complete action lattice \mathbf{A} , in order to reason about imperative programs interpreted as “single-flow” computations.

The results were obtained in collaboration with the following people: Alexandre Madeira, Luís Soares Barbosa, Mario Benevides and Manisha Jain, and are reported in the following publications:

- [GMB17] Gomes L., Madeira A., Barbosa L.S. On Kleene Algebras for Weighted Computation. In: Cavalheiro S., Fiadeiro J. (eds) Formal Methods: Foundations and Applications. SBMF 2017. Lecture Notes in Computer Science, vol 10623. Springer, Cham. (2017)
- [GMJB19] Gomes L., Madeira A., Jain M., Barbosa L.S. On the Generation of Equational Dynamic Logics for Weighted Imperative Programs. In: Ait-Ameur Y., Qin S. (eds) Formal Methods and Software Engineering. ICFEM 2019. Lecture Notes in Computer Science, vol 11852. Springer, Cham. (2019)
- [GMB19] Gomes, L., Madeira, A., and Barbosa, L. S. Generalising KAT to verify weighted computations. Sci. Ann. Comp. Sci. 29(2): 141-184. (2019)

Part 2: *weighted “multi-flow” computations*

Part 2 is concerned about weighted “multi-flow” computations. The approach is to define, first, an algebra to model programs of this class, leading to *weighted binary multirelations*. Next we define a formal semantics and a family of logics to reason about imperative programs interpreted as weighted “multi-flow” computations. The following list enumerates the main contributions.

1. (Chapter 6) An *algebra* to interpret programs, based on the concept of *weighted binary multirelation* (Definition 6.2.1).
2. (Chapter 7) The *semantics* of this class of programs, over an equational signature, function and program symbols, parametric on a *complete right residuated lattice* \mathbf{L} . In particular, programs are interpreted as *weighted binary multirelations*, to model a parallel execution leading from a state to a set of states.
3. (Chapter 8) A method for generating a family of **-free dynamic logics*, with semantics presented in Chapter 7 and the satisfaction relation defined also on a complete right residuated lattice \mathbf{L} . We call such logics $\Omega(\mathbf{L})$.

The results were obtained in collaboration with the following people: Alexandre Madeira and Luís Soares Barbosa, and are reported in the following papers:

- [Gom20] Gomes, L. On the construction of multi-valued concurrent dynamic logics. 2nd DaLi Workshop - Dynamic Logic: New Trends and Applications - Coallocated with 3rd World Congress on Formal Methods 2019. Lecture Notes in Computer Science, Springer. (2019)
- Gomes, L., Madeira, A., Barbosa, L.S. (2021) A semantics and a logic for *Fuzzy Arden Syntax*. Soft Computing. 25. 6789-6805. (2021)

One important feature which pervades the whole thesis is the parametric nature of the approach. Both semantics and logics (built over such semantics) are parametric on an algebraic structure able to model both computations and truth spaces in a weighted context. Such a structure is a complete action lattice in Section 3.5 and chapters 4 and 5. The instances presented for GKAT and I-GKAT, in Chapter 3, and in chapters 6, 7 and 8 rely on a complete right residuated lattice, a reduct of a complete action lattice without the operator $*$. We require completeness for both structures to guarantee the existence of arbitrary suprema, which will be relevant to prove the most important results of this thesis.

Differences from the literature

Weighted “single-flow” computations

The notion of weighted “single-flow” computation goes beyond the scope of probabilistic programs like (1). In this thesis, both semantic structures of weighted “single-flow” computations differs from the probabilistic scenario in the following sense:

- Our approach is not first and foremost syntactic, but rather mainly semantic, in the sense that we do not add new operators to algebraic structures or logics already established in the literature. We focus instead on providing weighted semantics to programs and predicates which live in those algebras and logics. Such approach is visible in Chapter 3, where the algebra used to formalise predicates is relaxed to capture a variety of weighted computations. Also in Chapter 4, although the syntax is the same as for classic imperative programs, the semantics of the language interprets assignments as weighted “single-flow” actions and predicates as weighted assertions, which define how much of a computation actually flows.
- We do not require, as it naturally occurs in a probabilistic event, that a single output is always obtained in each run of a program (in case of termination). In other words, the weights of all computations leaving a certain state do not necessarily sum to 1, i.e. we do not assure that an output is obtained in each run of a program.

Another relevant comparison relates to the research line that started with reference [DMS06], which presents a powerful framework to include programs and tests in the same formalisation, the KAD. Like

GKAT and I-GKAT introduced in Chapter 3, the authors also include relaxed versions of KAD, a structure based on Kleene algebra, to capture some notion of weight [DS11] and fuzziness [DM14]. The main features which distinguish both approaches are detailed below:

- The main difference between the structures of [DS08] and our approach lies on the construction of the structure itself: while ours is purely propositional and based on KAT, the one of [DS08] makes use of a unary operator, called the domain operator, to axiomatise the test algebra, resulting in a one sorted structure.
- Reference [DM14] goes even further by investigating a generalisation of these domain algebras to support fuzzy relations, taken as functions from pairs of elements to the interval $[0, 1]$. Differently from our approach, the authors study an axiomatisation of domain and codomain operators in the setting of IL-semirings. However, as shown in Chapter 3, we adopt a presentation similar to KAT when relaxing its Boolean subalgebra to obtain GKAT and I-GKAT. That results, for the purpose of this work, in a more direct comparison between the two obtained structures, either axiomatically and in terms of achieved results.

Weighted “multi-flow” computations

The mathematical formalisations that we propose in this thesis for weighted “multi-flow” computations also feature some clear differences with respect to other similar computational paradigms [MNM16, MRS13], as explained below:

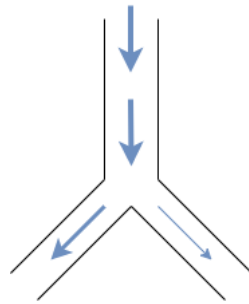


Figure 6: Conditional in a fuzzy programming language, illustrated by a liquid flowing through a ‘Y-shaped’ pipe.

- Probabilistic: we may distinguish our approach from the probabilistic paradigm, depicted in Figure 5, by comparing it with Figure 6, an intuitive metaphor to weighted “multi-flow” computations. We can observe that the liquid, represented by blue arrows, reaches a point where it flows through both channels in parallel (“multi-flow”), with different quantities of liquid going through each channel, represented by different thicknesses of arrows (weights). The behaviour of the probabilistic scenario is associated with a choice from a set of possible output states in a probability distribution. As a consequence, in each run of a probabilistic program we may obtain a different output (e.g. coin toss of Figure 5). In this sense, we may think of a probabilistic program as a set of relations between

states $\{(a, b)^\alpha, (a, c)^{1-\alpha}\}$, where α and $1 - \alpha$ are the probabilities of reaching, from state a , states b and c , respectively. In another direction, a weighted “multi-flow” computation adopts the behaviour metaphorically illustrated in Figure 6, where multiple instructions are executed in parallel with some weight associated. In this paradigm, from a single state a , the output of each run of a program is a single weighted set of output states b and c , with weights α and β , formally the pair $\{(a, \{b^\alpha, c^\beta\})\}$. The execution of a weighted “multi-flow” computation is deterministic in this sense.

- Weighted nondeterminism: we compare our approach with reference [MNM16], which proposes a method to generate formal semantics and logics for weighted programs. In such paper, despite the weighted nature of both programs and predicates, **if-then-else** statements are encoded by the $+$ operator of Kleene algebra, which is interpreted as nondeterministic choice. Moreover, the semantics of those programs is given by an algebra of square matrices, which relates to the semantics of programs introduced in **Part 1**.
- Weighted parallelism: our approach can also be compared with reference [MRS13], which incorporates a weight into the execution of programs running in parallel. However, the weight is introduced in the nondeterministic choice, and does not capture the parallelism inherent to weighted “multi-flow” computations.

In sum, the main differences to these paradigms reside in the conflict nondeterministic choice/parallelism, in the following way: the “flow” of the kind of computations introduced in **Part 2** does not live neither in a nondeterministic, nor in a probabilistic choice, arising instead in the very interpretation of a program as a weighted binary multirelation. Our thesis is that this interpretation formalises the weighted parallel behaviour illustrated in Section 1.3. Chapter 8 provides the mathematical framework, i.e. formal semantics which formalises programs with such behaviour, and Chapter 8 delineates a family of dynamic logics for their verification.

A comparison between our approach and variants of the concept of binary multirelation is in order. Although this thesis defines weighted binary multirelations as $\mathbf{2}^{W \times L^W}$ to model weighted “multi-flow” computations, alternative formalisations with distinct meanings could be adopted, as detailed below:

- One example is to define binary multirelation as $\mathbf{L}^{W \times \mathbf{2}^W}$, which adds weights to the transitions between a state and a set of states, as depicted in Figure 7. That corresponds to assigning an “external” weight to the transition itself (e.g. a failure in the run of a program).
- Another variant, $\mathbf{L}^{W \times L^W}$, depicted in Figure 8, not only considers this “external” weight, but incorporates as well the “internal” weight of the definition adopted in this thesis.

We finalise this subsection with Table 2, containing the frameworks introduced in this thesis marked with blue-coloured X marks.

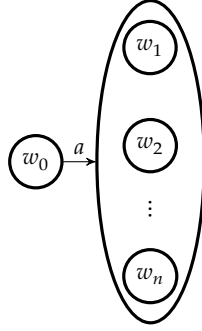


Figure 7: Illustration of $\mathbf{L}^{W \times 2^W}$

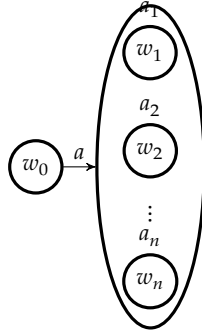


Figure 8: Illustration of $\mathbf{L}^{W \times \mathbf{L}^W}$

A note on syntax and notation

Programs in the two classes discussed in this thesis are described by two imperative programming languages: \mathcal{F}_1 for programs interpreted as weighted “single-flow” computations and \mathcal{F}_2 for programs interpreted as weighted “multi-flow” computations. The syntax of \mathcal{F}_1 -programs is given by the following grammar:

$t ::= c \mid x \mid f(t_1, \dots, t_n)$	functional terms
$p ::= p(t_1, \dots, t_n)$	predicate terms
$\pi_0 ::= \mathbf{skip} \mid x := t$	atomic programs
$\pi ::= \pi_0 \mid \pi; \pi \mid \mathbf{if } p \mathbf{ then } \pi \mathbf{ else } \pi$	compound programs
$\mathbf{while } p \mathbf{ do } \pi$	

Table 3: Syntax of \mathcal{F}_1

On the other hand the syntax of \mathcal{F}_2 -programs is given by:

	Prop	Eq	Rel	MRel	B	W
Hoare Logic (HL) [Hoa69]		X	X		X	
Propositional HL (PHL) [Koz00]	X		X		X	
Predicate transformer (wlp) [Dij76]	X	X	X		X	
Propositional Dynamic Logic (PDL) [FL77]	X		X		X	
First-order DL [HKT00]		X	X		X	
Kleene Algebra with Tests (KAT) [Koz97]	X		X		X	
Kleene Algebra with Domain (KAD) [DMS06]	X		X		X	
Probabilistic HL [dHdV02]		X	X			X
Probabilistic wlp [MMS96]		X	X			X
Probabilistic PDL [Koz85]	X		X			X
Probabilistic KAT [QWWG08]	X		X			X
Weighted PDL ($\mathcal{GDL}(\mathbf{A})$) [MNM16]	X		X			X
GKAT /I-GKAT [GMB19]	X		X			X
Weighted PHL [GMB17]	X		X			X
Weighted Equational Dynamic Logic ($\Gamma(\mathbf{A})$) [GMJB19]		X	X			X
“Weighted” KAD [DS11]	X		X			X
Algebra of binary multirelations [Rew03]	X			X	X	
Concurrent PDL (CPDL) [Pel87]	X			X	X	
Concurrent KAT (CKAT) [JM16]	X		X		X	
Concurrent Dynamic Algebra [FS15]	X			X	X	
Probabilistic CKA [MRS13]	X		X			X
Algebra of probabilistic multirelations [Tsu12]	X			X		X
Algebra of weighted binary multirelations	X			X		X
Weighted CPDL [Gom20]	X			X		X
Weighted “multi-flow” dynamic logic ($\Omega(\mathbf{L})$)		X		X		X

Table 2: Taxonomy of related work and the frameworks introduced in this thesis

$t ::= c \mid x \mid f(t_1, \dots, t_n)$	functional terms
$p ::= \rho(t_1, \dots, t_n)$	predicate terms
$\pi_0 ::= \mathbf{skip} \mid x := t$	atomic programs
$\pi ::= \pi_0 \mid \pi; \pi \mid \mathbf{if } p \mathbf{ then } \pi_1 \mathbf{ else } \pi_2$	compound programs
$\mathbf{if } p \mathbf{ then } \pi_1 \mathbf{ else } \pi_2 \mathbf{ aggregate} \mid$	
$\mathbf{if } p \mathbf{ then } \pi_1 \mathbf{ else } \pi_2 \mathbf{ aggregate defuzzify}$	

Table 4: Syntax of \mathcal{F}_2

where

- $x \in X$ are *variables*;

- $f \in F$ are *function symbols*. $(F_n)_{n \in \mathbb{N}_0} \subseteq F$ denotes sets of function symbols with arity n . Symbols $c \in F_0$ are called constants. Function symbols are interpreted in F as $f(t_1, \dots, t_n) : \mathbb{R} \times \dots \times \mathbb{R} \rightarrow \mathbb{R}$ (e.g. $+$, $\sqrt{\cdot}$);
- $p \in P$ are *predicate symbols*. $(P_n)_{n \in \mathbb{N}_0} \subseteq P$ denotes sets of predicate symbols with arity n . Predicate symbols are interpreted in P as $p(t_1, \dots, t_n) : \mathbb{R}^n \rightarrow \mathbf{2}$ (e.g. $=$, \geq).

Additionally, notation $T(X)$ stands for the set of terms with variables in X , and $T_F(X)$ (respectively, $T_P(X)$) represents its restriction to functional (respectively, predicate) terms. We denote by Prog_0 the set of all *atomic* programs over X , i.e. the set of all assignments of terms in $T_F(X)$ to variables in X

$$\text{Prog}_0 = \{x := t \mid x \in X \text{ and } t \in T_F(X)\}$$

Moreover, the set of (compound) programs generated with the grammars described above is named Prog .

The grammars defined above mean that a programming assignment of values from a data space to a variable is taken as the elementary construction, with terms being defined over a signature of program variables, predicate and function symbols. Although the syntax of \mathcal{F}_1 and \mathcal{F}_2 are similar to the one typical of classic imperative programming languages, their semantics, as we have seen in Subsection 1.2, is given in terms of “weighted” computations, and thus both terms, predicates and programs are interpreted accordingly.

Finally, the text adopts the following notation:

- Logic formulas will be denoted as $\rho_i, i \in \mathbb{N}$.
- Programs will be denoted as $\pi_i, i \in \mathbb{N}$.
- Weighted sets will often be denoted as φ, ψ .
- Weighted relations will often be denoted as μ, ν, ζ .
- Weighted languages will often be denoted as $\lambda_i, i \in \mathbb{N}$.
- Both a lattice itself and its support set will be denoted by the same letter interchangeably, as **A, L**.
- Program states will often be denoted as $w_i, i \in \mathbb{N}$.
- Both the expressions in the logic and the least (greatest) element of the lattice will be denoted as \perp (\top).
- The abbreviation *iff* stands, as usual, for *if and only if*.

 BACKGROUND

This chapter surveys some important concepts that will set forth both the algebraic and logic constructions presented along the thesis. These includes the notions of Kleene algebra, action lattice, and variants of dynamic logic. In order to give meaning to weighted computations, we present some instances of the semantic structures and dynamic logics based on the theory of fuzzy sets and fuzzy relations [Zad65].

2.1 Propositional dynamic logic

We start by recalling PDL [HKT00], namely its syntax, semantics and satisfaction relation.

Syntax. The language of PDL is built over *atomic programs* and *propositions*. The set of all atomic programs is denoted by Prog_0 and the set of atomic propositions by Prop .

The set of all compound programs, denoted by Prog , and the set of all PDL formulas, denoted by P , are built inductively from the atomic ones, by the grammars

$$\pi ::= \pi_0 \mid \pi; \pi \mid \pi \cup \pi \mid \pi^* \mid \rho?$$

for $\pi_0 \in \text{Prog}_0$ and $\rho \in P$, and

$$\rho ::= \perp \mid p \mid \rho \rightarrow \rho \mid [\pi]\rho$$

for $p \in \text{Prop}$ and $\pi \in \text{Prog}$, respectively.

Compound programs and propositions have the following intuitive meanings:

$[\pi]\rho$ “Necessarily after the execution of π , ρ is true.”

$\pi_1; \pi_2$ “Execute π_1 , then execute π_2 .”

$\pi_1 \cup \pi_2$ “Choose π_1 or π_2 nondeterministically.”

π^* “Execute π a nondeterministically finite number of times.”

$\rho?$ “Test ρ ; proceed if true, fail otherwise”.

Other conventional operators for formulas can be formed from the ones presented above, as in classic propositional logic: \wedge , \vee , \neg and \top . The modal operator $\langle \rangle$ is the dual of $[\]$, and is defined as

$\langle \pi \rangle \rho = \neg [\pi] \neg \rho$. Its intuitive meaning is “There is a computation of π that terminates in a state satisfying ρ ”.

The expressiveness of PDL allows to abstract the following programming constructs:

$$\begin{aligned} \mathbf{skip} &\stackrel{\text{def}}{=} \top? \\ \mathbf{fail} &\stackrel{\text{def}}{=} \perp? \\ \mathbf{if } \rho \mathbf{ then } \pi_1 \mathbf{ else } \pi_2 &\stackrel{\text{def}}{=} \rho?; \pi_1 \cup \neg \rho?; \pi_2 \\ \mathbf{while } \rho \mathbf{ do } \pi &\stackrel{\text{def}}{=} (\rho?; \pi)^*; \neg \rho? \\ \{\rho_1\} \pi \{\rho_2\} &\stackrel{\text{def}}{=} \rho_1 \rightarrow [\pi] \rho_2 \end{aligned}$$

The programs **skip** and **fail** are, respectively, the program that does nothing and the one that fails. The **if-then-else** and **while-do** operators are the usual *conditional* and *while loop* constructs found in conventional programming languages. Finally PDL allows to encode the Hoare partial correctness assertion $\{\rho_1\} \pi \{\rho_2\}$.

Semantics.

The semantics of PDL is obtained from modal logic. Programs and propositions are interpreted over a *Kripke frame* $\mathfrak{K} = (K, m_{\mathfrak{K}})$, where K is a set of states and $m_{\mathfrak{K}}$ is a function assigning a subset of K to each atomic proposition in P_0 and a binary relation to each atomic program in Prog_0 , as follows

$$\begin{aligned} m_{\mathfrak{K}}(\pi_0) &\subseteq K \times K, \text{ for } \pi_0 \in \text{Prog}_0 \\ m_{\mathfrak{K}}(p) &\subseteq K, \text{ for } p \in \text{Prop} \end{aligned}$$

The function $m_{\mathfrak{K}}$ can be inductively extended to give meaning to all elements of P and Prog .

$$\begin{aligned} m_{\mathfrak{K}}(\pi) &\subseteq K \times K, \text{ for } \pi \in \text{Prog} \\ m_{\mathfrak{K}}(\rho) &\subseteq K, \text{ for } \rho \in P \end{aligned}$$

Intuitively, the set $m_{\mathfrak{K}}(\rho)$ is the set of states satisfying ρ and $m_{\mathfrak{K}}(\pi)$ represents the semantics of program π as a relation between states.

The meanings of compound programs and propositions can be defined, as well, by mutual induction on the structure of ρ and π . Let operator \circ be the relational composition, operator $*$ the reflexive transitive closure on binary relations, $\Delta_{m_{\mathfrak{K}}(\rho)}$ the coreflexive correspondent to the set $m_{\mathfrak{K}}(\rho)$ and $\mathbb{P}(\Pi_1)$ the extension of the projection Π_1 to a set.

$$\begin{aligned}
m_{\mathfrak{R}}(\perp) &\stackrel{\text{def}}{=} \emptyset \\
m_{\mathfrak{R}}(\rho_1 \rightarrow \rho_2) &\stackrel{\text{def}}{=} (K \setminus m_{\mathfrak{R}}(\rho_1)) \cup m_{\mathfrak{R}}(\rho_2) \\
m_{\mathfrak{R}}([\pi]\rho) &\stackrel{\text{def}}{=} \mathbb{P}(\Pi_1)(\Delta_{m_{\mathfrak{R}}(\rho)} \circ m_{\mathfrak{R}}(\pi)) \\
&= \{w_1 \mid \forall_{w_2 \in K} \text{ if } (w_1, w_2) \in m_{\mathfrak{R}}(\pi) \text{ then } w_2 \in m_{\mathfrak{R}}(\rho)\} \\
m_{\mathfrak{R}}(\pi_1; \pi_2) &\stackrel{\text{def}}{=} m_{\mathfrak{R}}(\pi_2) \circ m_{\mathfrak{R}}(\pi_1) \\
&= \{(w_1, w_2) \mid \exists_{w' \in K} \cdot (w_1, w') \in m_{\mathfrak{R}}(\pi_1) \text{ and } (w', w_2) \in m_{\mathfrak{R}}(\pi_2)\} \\
m_{\mathfrak{R}}(\pi_1 \cup \pi_2) &\stackrel{\text{def}}{=} m_{\mathfrak{R}}(\pi_1) \cup m_{\mathfrak{R}}(\pi_2) \\
m_{\mathfrak{R}}(\pi^*) &\stackrel{\text{def}}{=} m_{\mathfrak{R}}(\pi)^* = \bigcup_{n \geq 0} m_{\mathfrak{R}}(\pi)^n \\
m_{\mathfrak{R}}(\rho?) &\stackrel{\text{def}}{=} \{(u, u) \mid u \in m_{\mathfrak{R}}(\rho)\}
\end{aligned}$$

Satisfaction. The satisfaction relation is defined as

$$\begin{aligned}
w &\not\models \perp \\
w &\models p \stackrel{\text{def}}{\Leftrightarrow} w \in m_{\mathfrak{R}}(p) \\
w &\models \rho_1 \rightarrow \rho_2 \stackrel{\text{def}}{\Leftrightarrow} w \models \rho_1 \text{ implies } w \models \rho_2 \\
w &\models [\pi]\rho \stackrel{\text{def}}{\Leftrightarrow} \forall w' \text{ if } (w, w') \in m_{\mathfrak{R}}(\pi) \text{ then } w' \models \rho
\end{aligned}$$

and the operators defined for programs and propositions have the following meaning

$$\begin{aligned}
m_{\mathfrak{R}}(\rho_1 \wedge \rho_2) &\stackrel{\text{def}}{=} m_{\mathfrak{R}}(\rho_1) \cap m_{\mathfrak{R}}(\rho_2) \\
m_{\mathfrak{R}}(\rho_1 \vee \rho_2) &\stackrel{\text{def}}{=} m_{\mathfrak{R}}(\rho_1) \cup m_{\mathfrak{R}}(\rho_2) \\
m_{\mathfrak{R}}(\neg\rho) &\stackrel{\text{def}}{=} K \setminus m_{\mathfrak{R}}(\rho) \\
m_{\mathfrak{R}}(\top) &\stackrel{\text{def}}{=} K \\
m_{\mathfrak{R}}(\langle\pi\rangle\rho) &\stackrel{\text{def}}{=} \Delta_{m_{\mathfrak{R}}(\rho)} \circ m_{\mathfrak{R}}(\pi) \\
&= \{w \mid \exists_{w' \in K} \cdot (w, w') \in m_{\mathfrak{R}}(\pi) \text{ and } w' \in m_{\mathfrak{R}}(\rho)\}
\end{aligned}$$

The following example, from [HKT00], provides a simple illustration of a Kripke frame, and discusses the validity of some PDL formulas in the model.

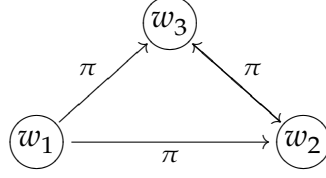
Example 2.1.1. Let p be an atomic proposition, π an atomic program and $\mathfrak{R} = (K, m_{\mathfrak{R}})$ a Kripke frame where

$$K = \{w_1, w_2, w_3\}$$

$$m_k(p) = \{w_1, w_2\}$$

$$m_k(\pi) = \{(w_1, w_2), (w_1, w_3), (w_2, w_3), (w_3, w_2)\}.$$

as illustrated below



In this model, for instance, $w_1 \models \langle \pi \rangle \neg p \wedge \langle \pi \rangle p$, $w_2 \models [\pi] \neg p$ and $w_3 \models [\pi] p$.

Note that PDL programs are built using the primitive operators $;$, \cup and $*$, which are inherited from regular expressions. Thus, those programs can be viewed as regular expressions over atomic programs and tests. In fact, such connection makes Kleene algebra the *de facto mathematical structure* to abstract PDL programs, paving the way to become the standard computational model for many current frameworks for reasoning about programs. Along this thesis, we resort most of the times to Kleene algebra and variants as an algebra for the kind of computation discussed. Next section recalls them.

2.2 Kleene algebra and action lattice

These structures will play a fundamental role in the frameworks introduced in the thesis. First, we present Kleene algebra, established as the *de facto algebra* for abstract imperative programs. Next, the notion of action lattice is introduced as a Kleene algebra enriched with the structure of a residuated lattice, which offers a proper truth space for dynamic logic constructions.

Definition 2.2.1 (Kleene algebra). *A Kleene algebra is a tuple*

$$(K, +, ;, *, 0, 1)$$

where K is a set, $+$, $;$ are binary operators, $*$ is a unary operator, and $0, 1$ are constants, and a partial order \leq defined by $a \leq b$ if and only if $a + b = b$, verifying the following axioms:

$$p + (q + r) = (p + q) + r \quad (4)$$

$$p + q = q + p \quad (5)$$

$$p + p = p \quad (6)$$

$$p + 0 = p \quad (7)$$

$$p; (q; r) = (p; q); r \quad (8)$$

$$p;1 = 1;p = p \quad (9)$$

$$p;(q+r) = (p;q) + (p;r) \quad (10)$$

$$(p+q);r = (p;r) + (q;r) \quad (11)$$

$$p;0 = 0;p = 0 \quad (12)$$

$$1 + p;p^* = p^* \quad (13)$$

$$1 + p^*;p = p^* \quad (14)$$

$$q + p;r \leq r \Rightarrow p^*;q \leq r \quad (15)$$

$$q + r;p \leq r \Rightarrow q;p^* \leq r \quad (16)$$

In program semantics, operator $+$ is interpreted as nondeterministic choice, $;$ as sequential composition and $*$ as iteration.

The definition and axiomatisation of an action lattice are now recalled below.

Definition 2.2.2 (Action lattice [Koz94b]). *An action lattice is a tuple*

$$(A, +, ;, *, \rightarrow, \cdot, 0, 1)$$

satisfying the axioms characterising a Kleene algebra, and additionally the following ones

$$a;b \leq c \Leftrightarrow b \leq a \rightarrow c \quad (17)$$

$$(a \rightarrow a)^* = a \rightarrow a \quad (18)$$

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c \quad (19)$$

$$a \cdot b = b \cdot a \quad (20)$$

$$a \cdot a = a \quad (21)$$

$$a + (a \cdot b) = a \quad (22)$$

$$a \cdot (a + b) = a \quad (23)$$

Operators $+$ and \cdot represent the supremum and the infimum w.r.t. \leq , respectively. Note that $+$ has a double role, depending on the interpretation over a Kleene algebra or an action lattice. We say that an action lattice \mathbf{A} is *complete* when every subset of its carrier A has both supremum and infimum with respect to \leq . The greatest and least elements are denoted in the sequel by \top and \perp , respectively. Consider a set I . We say that \mathbf{A} is *linear* if it satisfies, for any set $\{a_i \mid i \in I\}$, the property

$$\sum_{i \in I} a_i = a_j, \text{ for some } j \in I \quad (24)$$

The following two lattices are examples of linear action lattices.

Example 2.2.1. (**2** - the Boolean lattice). *The first example is the Boolean lattice*

$$2 = (\{\top, \perp\}, \vee, \wedge, *, \rightarrow, \wedge, \perp, \top)$$

with the standard interpretation of Boolean connectives. Operator $*$ maps each element to \top , and \rightarrow corresponds to logical implication.

Example 2.2.2. (**3** - the three-valued lattice). *The second example is provided by the three-element lattice, which introduces an explicit denotation u for “unknown” (or “undefined”).*

$$3 = (\{\top, u, \perp\}, \vee, \wedge, *, \rightarrow, \wedge, \perp, \top)$$

where

\vee	\perp	u	\top
\perp	\perp	u	\top
u	u	u	\top
\top	\top	\top	\top

\wedge	\perp	u	\top
\perp	\perp	\perp	\perp
u	\perp	u	u
\top	\perp	u	\top

\rightarrow	\perp	u	\top
\perp	\top	\top	\top
u	\perp	\top	\top
\top	\perp	u	\top

$*$	\perp	\top
\perp	\perp	\top
u	u	\top
\top	\top	\top

An action lattice is called \mathbb{I} -*action lattice* when the identity of operator $;$ coincides with the greatest element of the residuated lattice, i.e. $1 = \top$. It is called \mathbb{H} -*action lattice* if operator $;$ coincides with the infimum of the residuated lattice, i.e. for any $a, b \in A$, $a; b = a \cdot b$. For a complete action lattice \mathbf{A} , since operators $+$, $;$ and \cdot are associative and have identity, they admit a n -ary iterated version, represented by \sum , \prod and \bigwedge , respectively.

Beyond the lattices of examples 2.2.1 and 2.2.2, the following are examples of complete action lattices.

Example 2.2.3. (2^S - powerset of S). *For a fixed, finite set S , another instance of is*

$$2^S = (P(S), \cup, \cap, *, \rightarrow, \cap, \emptyset, S)$$

where $P(S)$ denotes the powerset of S , \cup and \cap are set union and intersection, respectively, $*$ maps each set $X \in P(S)$ into S , and $X \rightarrow Y = X^C \cup Y$, where $X^C = \{x \in S \mid x \notin X\}$.

Example 2.2.4. (\mathbf{L} - Łukasiewicz arithmetic lattice).

$$\mathbf{L} = ([0, 1], \max, \odot, *, \rightarrow, \min, 0, 1)$$

where $x \rightarrow y = \min\{1, 1 - x + y\}$, $x \odot y = \max\{0, x + y - 1\}$ and $*$ maps each point of the interval $[0, 1]$ to 1, and \odot maps each point of the interval $[0, 1]$ to 0.

Example 2.2.5. (The Floyd-Warshall algebra).

$$\mathbb{N}_{\perp, \top}^+ = (\{\perp, 0, 1, \dots, \top\}, \max, +, *, \smile, \min, \perp, 0)$$

where $+$ extends addition on \mathbb{N} by considering \perp as its absorbent element and $a + \top = \top = \top + a$ for any $a \neq \perp$. Operation \max (respectively, \min) is defined as the maximum (respectively, minimum) under the order $\perp < 0 < \dots < \top$. Operation \smile is truncated subtraction

$$a \smile b = \begin{cases} \top, & \text{if } a = \perp \text{ or } b = \top \\ b - a, & \text{if } b \geq a \text{ and } a, b \in \mathbb{N} \\ 0, & \text{if } a > b \text{ and } a, b \in \mathbb{N} \\ \perp & \text{otherwise} \end{cases}$$

$*$	
\perp	0
0	0
i	\top
\top	\top

and, for any natural $i > 0$,

Example 2.2.6. (**P** - the product algebra).

$$\mathbf{P} = ([0, 1], \max, \cdot, *, \rightarrow, \min, 0, 1)$$

where \cdot is the usual multiplication of real numbers,

$$x \rightarrow y = \begin{cases} 1, & \text{if } x \leq y \\ y/x, & \text{if } y < x \end{cases}$$

$/$ is real division and $*$ maps each point of the interval $[0, 1]$ to 1.

Example 2.2.7. (**G** - the Gödel algebra). Gödel algebras are the locally finite variety of Heyting algebras. Formally,

$$\mathbf{G} = ([0, 1], \max, \min, *, \rightarrow, \min, 0, 1)$$

where

$$x \rightarrow y = \begin{cases} 1, & \text{if } x \leq y \\ y, & \text{if } y < x \end{cases}$$

Example 2.2.8. (**REL**(X) - relational algebra over a set X). Let us consider the action lattice defined by relations over a set X . The corresponding Kleene algebra turns to be quite paradigmatic, since it underlies

most standard semantics for sequential programs based on input/output relations. Then, given a set X , we have

$$\mathbf{REL}(X) = (\mathcal{P}(X^2), \cup, \circ, *, \cdot, \cap, \emptyset, \Delta)$$

where \cup and \cap stand for set union and intersection, respectively, \emptyset represents the empty relation and Δ the diagonal relation $\{(x, x) | x \in X\}$. Operation $*$ is Kleene closure, recursively defined, for each $R \in \mathcal{P}(X^2)$, by $R^* = \bigcup_{n \leq \omega} R^n$, where $R^0 = \Delta$ and $R^{n+1} = R^n \circ R$. Finally the residuum is given by $Q \cdot R = \{(x, y) | \text{for every } z \text{ if } (y, z) \in Q \text{ then } (x, z) \in R\}$.

Example 2.2.9. ($\mathbf{LAN}(\Sigma)$ - languages over an alphabet Σ). Let us consider the action lattice defined by the finite languages on a finite alphabet Σ . Then, for a given finite alphabet Σ , we define the action lattice of languages over Σ as

$$\mathbf{LAN}(\Sigma) = (\mathcal{P}(\Sigma^*), \cup, \cdot, *, \rightarrow, \cap, \emptyset, \{\epsilon\})$$

where \cup and \cap stand for set union and intersection, \emptyset represents the empty language, ϵ is the empty word, the operation $*$ is the Kleene star defined by $L^* = \bigcup_{n \geq 0} L^n = \{w_1 \cdots w_n | w_i \in L, 1 \leq i \leq n\}$ and $L^0 = \{\epsilon\}$ and $L^{n+1} = L \cdot L^n$. The composition \cdot is defined by $L_1 \cdot L_2 = \{w_1 \cdot w_2 | w_1 \in L_1 \text{ and } w_2 \in L_2\}$ and the residuum \rightarrow by $L_1 \rightarrow L_2 = \{w_2 | \forall w_1 \cdot \text{if } w_1 \in L_1 \text{ then } w_1 \cdot w_2 \in L_2\}$.

Example 2.2.10. (\mathbf{W}_k - finite Wajsberg hoops). Let us consider now an action lattice endowing the finite Wajsberg hoop with a star operator [BF00]. For a fixed natural k and a generator a , one gets

$$W_k = (W_k, +, ;, *, \rightarrow, \min, 0, 1)$$

where $W_k = \{a^0, a^1, \dots, a^{k-1}\}$, $1 = a^0$ and $0 = a^{k-1}$. Moreover, for any $m, n \leq k-1$, $a^m + a^n = a^{\min\{m, n\}}$, $a^m ; a^n = a^{\min\{m+n, k-1\}}$, $(a^m)^* = a^0$ and $a^m \rightarrow a^n = a^{\max\{n-m, 0\}}$.

Example 2.2.11. (\mathbb{R} - the tropical algebra). Finally, the $(\min, +)$ Kleene algebra [Koz92], known as the tropical semiring, can be extended to an action lattice through the introduction of residuation \rightarrow :

$$\mathbf{R} = (\mathbb{R}_0^+ \cup \{+\infty\}, \min, +_{\mathbf{R}}, *, \rightarrow, \min, +\infty, 0_{\mathbf{R}})$$

where, for any $x, y \in \mathbb{R}_0^+ \cup \{+\infty\}$, $x^* = 0_{\mathbf{R}}$, $x \rightarrow y = \max\{y - x, 0\}$, and $+_{\mathbf{R}}, 0_{\mathbf{R}}$ are the sum of real numbers and the real number 0, respectively, with $\mathbb{R}_0^+ = \{x \in \mathbb{R} | x \geq 0\}$.

The framework introduced in **Part 2** resorts, however, to a restriction of an action lattice without the operator $*^1$, i.e.

Definition 2.2.3 (Right residuated lattice). A right residuated lattice ² is a tuple

$$\mathbf{L} = (L, +, ;, \rightarrow, \cdot, 0, 1)$$

- 1 The class of programs that we address in **Part 2** do not include loops, and thus the lattice which acts as the parameter to model computations does not need to have an operator to model iteration. That is the reason we opt for a more simple structure.
- 2 Note that while the most common nomenclature for \leftarrow (\rightarrow) is *right* (*left*) *division*, which comes from the definition of *residuated lattice* [GJK007], we follow reference [Koz94b], where \leftarrow (\rightarrow) is called the *left* (*right*) *residual*.

where L is a set, $0, 1$ are constants, and $+, ;, \rightarrow$ and \cdot are binary operations over L satisfying axioms (4)-(12), (17) and (19)-(23).

The order relation \leq is the same as for an action lattice. Moreover, the nomenclatures *complete*, *linear*, \mathbb{I} and \mathbb{H} are maintained for right residuated lattices. Naturally, all the examples of action lattices presented above (2.2.1 - 2.2.11) can be reduced to right residuated lattices, by disregarding operator $*$.

2.3 Generating multi-valued propositional dynamic logics

This section illustrates the method introduced in [MNM16], for generation of multi-valued dynamic logics, parametric on an action lattice. The signature, formulæ, semantics and satisfaction are recalled below. Given an action lattice \mathbf{A} , the corresponding family of logics is denoted by $\mathcal{GDL}(\mathbf{A})$.

Signatures. A signature of $\mathcal{GDL}(\mathbf{A})$ is a pair

$$\Delta = (\text{Prog}_0, \text{Prop})$$

of atomic programs and propositions, respectively.

Formulæ. The *set of compound programs*, denoted by Prog , is generated by

$$\pi ::= \pi_0 \mid \pi; \pi \mid \pi + \pi \mid \pi^*$$

where $\pi_0 \in \text{Prog}_0$. Given a signature $\Delta = (\text{Prog}_0, \text{Prop})$, we define the $\mathcal{GDL}(\mathbf{A})$ -formulæ for Δ , denoted by $\text{Fm}^{\mathcal{GDL}(\mathbf{A})}(\Delta)$, as the ones generated by the grammar

$$\rho ::= \top \mid \perp \mid p \mid \rho \vee \rho \mid \rho \wedge \rho \mid \rho \rightarrow \rho \mid \rho \leftrightarrow \rho \mid \langle \pi \rangle \rho \mid [\pi] \rho$$

for $p \in \text{Prop}$ and $\pi \in \text{Prog}$. Note that this corresponds to the *positive* fragment of the propositional dynamic logic. Moreover, in order to support a multi-valued truth space, the negation is not explicitly denoted, being instead defined as $\rho \rightarrow \perp$, for $\rho \in \text{Fm}^{\mathcal{GDL}(\mathbf{A})}(\Delta)$. Hence, contrary to what occurs in *PDL*, where some operators are defined by abbreviation ($\vee, \top, \perp, \langle \rangle$), using the negation \neg , all those operators are included in the syntax of $\mathcal{GDL}(\mathbf{A})$.

Semantics. The first step is to introduce the space where the computations of $\mathcal{GDL}(\mathbf{A})$ are to be interpreted. Based on the classic matricial constructions over Kleene algebras (see [Con12, Koz94a]), let us define

$$\mathbb{M}_n(\mathbf{A}) = (M_n(\mathbf{A}), +, ;, *, \mathbf{0}, \mathbf{1})$$

as follows:

1. $M_n(\mathbf{A})$ is the space of $(n \times n)$ -matrices over \mathbf{A}
2. for any $A, B \in M_n(\mathbf{A})$, define $M = A+B$ by $M_{i,j} = A_{i,j} + B_{i,j}$, $i, j \leq n$.

3. for any $A, B \in M_n(\mathbf{A})$, define $M = A ; B$ by $M_{i,j} = \sum_{k=1}^n (A_{i,k} ; B_{k,j})$ for any $i, j \leq n$.

4. for any $M = [a] \in \mathbb{M}_1(\mathbf{A})$, $M^* = [a^*]$;

for any $M = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array} \right] \in M_n(\mathbf{A})$, $n > 1$, where A and D are square matrices, define

$$M^* = \left[\begin{array}{c|c} F^* & F^* ; B ; D^* \\ \hline D^* ; C ; F^* & D^* + (D^* ; C ; F^* ; B ; D^*) \end{array} \right]$$

where $F = A + B ; D^* ; C$. Note that this construction is recursively defined from the base case (where $n = 2$) where the operations of the base action lattice \mathbf{A} are used.

5. $\mathbf{1}$ and $\mathbf{0}$ are the $(n \times n)$ -matrices defined by $\mathbf{1}_{i,j} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$ and $\mathbf{0}_{i,j} = 0$, for any $i, j \leq n$.

The following classic result (e.g. [Con12, Koz94a]) establishes that Kleene algebras are closed under formation of matrices.

Theorem 2.3.1. *The structure $\mathbb{M}_n(\mathbf{A}) = (M_n(\mathbf{A}), +, ;, *, \mathbf{0}, \mathbf{1})$ defined above is a Kleene algebra.*

$\mathcal{GDL}(\mathbf{A})$ -models for a signature $\Delta = (\text{Prog}_0, \text{Prop})$, denoted by $\text{Mod}^{\mathcal{GDL}(\mathbf{A})}(\Delta)$, consists of tuples

$$\mathcal{A} = (W, V, (\mathcal{A}_\pi)_{\pi \in \text{Prog}_0})$$

where W is a finite set (of states), $V : \text{Prop} \times W \rightarrow \mathbf{A}$ is a function, and $\mathcal{A}_\pi \in M_n(\mathbf{A})$, with n standing for the cardinality of W .

The interpretation of programs in these models is done over the space of the matrices over the Kleene algebra of \mathbf{A} . Each matrix represents the effect of a program executing from any point of the model. Formally, the interpretation of a program $\pi \in \text{Prog}_0$ in a model $\mathcal{A} \in \text{Mod}^{\mathcal{GDL}(\mathbf{A})}(\Delta)$ is recursively defined, from the set of atomic programs $(\mathcal{A}_{\pi_0})_{\pi_0 \in \text{Prog}_0}$, as follows:

$$\mathcal{A}_{\pi ; \pi'} = \mathcal{A}_\pi ; \mathcal{A}_{\pi'} , \quad \mathcal{A}_{\pi + \pi'} = \mathcal{A}_\pi + \mathcal{A}_{\pi'} \quad \text{and} \quad \mathcal{A}_{\pi^*} = \mathcal{A}_\pi^* .$$

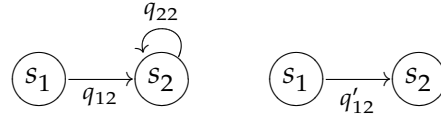
together with the constants' interpretation $\mathcal{A}_1 = \mathbf{1}$ and $\mathcal{A}_0 = \mathbf{0}$. Note that the adoption of the classic matricial constructions on Kleene algebras [Con12, Koz94a], where the operations are defined for $n \times n$ matrices, assumes the finiteness of the state spaces (since n stands for the cardinality of W). Although sum, composition and union can be easily generalised for the infinite setting, it is not clear if a similar construction exists for the star operation.

Let us now give some examples of this construction, by instantiating it with some of the action lattices enumerated above.

Example 2.3.1. Let us fix a complete action lattice $\mathbf{A} = (A, +, ;, 0, 1, *, \rightarrow, \cdot)$ and a signature $(\{\pi, \pi'\}, \{p\})$. Moreover, consider the model $\mathcal{A} = (W, V, (A_\pi)_{\pi \in \Pi})$, with $W = \{s_1, s_2\}$ and the following atomic programs

$$A_\pi = \begin{bmatrix} \perp & q_{12} \\ \perp & q_{22} \end{bmatrix} \quad A_{\pi'} = \begin{bmatrix} \perp & q'_{12} \\ \perp & \perp \end{bmatrix}$$

which can be represented by the following labelled transition systems:



Let $\mathbf{A} = \mathbf{2}$, from Example 2.2.1. Taking $q_{12} = q_{22} = q'_{1,2} = \top$ we get the standard adjacency matrices of the graph underlying the transition systems. In this case, we interpret choice $\pi + \pi'$ by

$$A_{\pi+\pi'} = A_\pi + A_{\pi'} = \begin{bmatrix} \perp & \top \\ \perp & \top \end{bmatrix} + \begin{bmatrix} \perp & \top \\ \perp & \perp \end{bmatrix} = \begin{bmatrix} \perp \vee \perp & \top \vee \top \\ \perp \vee \perp & \top \vee \perp \end{bmatrix} = \begin{bmatrix} \perp & \top \\ \perp & \top \end{bmatrix}$$

The interpretation of the composition $\pi; \pi'$ is as follows,

$$A_{\pi; \pi'} = \begin{bmatrix} \perp & \top \\ \perp & \top \end{bmatrix}; \begin{bmatrix} \perp & \top \\ \perp & \perp \end{bmatrix} = \begin{bmatrix} (\perp \wedge \perp) \vee (\top \wedge \perp) & (\perp \wedge \top) \vee (\top \wedge \perp) \\ (\perp \wedge \perp) \vee (\top \wedge \perp) & (\perp \wedge \top) \vee (\top \wedge \perp) \end{bmatrix}$$

$$= \begin{bmatrix} \perp & \perp \\ \perp & \perp \end{bmatrix}$$

Thus, as expected,

$$A_{\pi'; \pi} = \begin{bmatrix} \perp & \top \\ \perp & \perp \end{bmatrix}$$

For the interpretation of the π closure, we have

$$A_{\pi^*} = (A_\pi)^* \begin{bmatrix} \perp & \top \\ \perp & \top \end{bmatrix} = \begin{bmatrix} f^* & f^* \wedge \top \wedge \top^* \\ \top^* \wedge \perp \wedge \perp^* & \top^* \vee (\top^* \wedge \perp \wedge \top \wedge \top) \end{bmatrix}$$

where $f = \perp \vee (\top \wedge \top^* \wedge \perp) = \perp$; hence $A_{\pi^*} = \begin{bmatrix} \top & \top \\ \perp & \top \end{bmatrix}$.

Now, we introduce some uncertainty on the execution of the programs, to illustrate its effect when computing the sequential composition of two programs. For this purpose, we take $\mathbf{A} = \mathbf{3}$, from Example 2.2.2, as the parameter. Considering $q_{12} = q_{22} = \top$ and $q'_{12} = u$, we have

$$A_{\pi';\pi} = \begin{bmatrix} \perp & u \\ \perp & \perp \end{bmatrix}; \begin{bmatrix} \perp & \top \\ \perp & \top \end{bmatrix} = \begin{bmatrix} (\perp \wedge \perp) \vee (u \wedge \perp) & (\perp \wedge \top) \vee (u \wedge \top) \\ (\perp \wedge \perp) \vee (\perp \wedge \perp) & (\perp \wedge \top) \vee (\perp \wedge \top) \end{bmatrix} = \begin{bmatrix} \perp & u \\ \perp & \perp \end{bmatrix}$$

As expected, the unknown factor affecting transition $s_1 \rightarrow s_2$ in A'_π is propagated to transition $s_1 \rightarrow s_2$ in $A_{\pi';\pi}$.

If, nevertheless, a continuous space is required to define the “unknown” metric, the Łukasiewicz arithmetic lattice \mathbf{L} , from Example 2.2.4, would be a suitable option. Consider, for instance, $q_{12} = a$, $q_{22} = b$ and $q'_{12} = c$ for some $a, b, c \in [0, 1]$. In this case the compound program representing the nondeterministic choice $A_{\pi+\pi'}$ is computed as follows:

$$A_{\pi+\pi'} = \begin{bmatrix} 0 & a \\ 0 & b \end{bmatrix} + \begin{bmatrix} 0 & c \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} \max\{0, 0\} & \max\{a, c\} \\ \max\{0, 0\} & \max\{b, 0\} \end{bmatrix} = \begin{bmatrix} 0 & \max\{a, c\} \\ 0 & b \end{bmatrix}$$

The sequential composition is computed as follows.

$$A_{\pi';\pi} = \begin{bmatrix} 0 & c \\ 0 & 0 \end{bmatrix}; \begin{bmatrix} 0 & a \\ 0 & b \end{bmatrix} = \begin{bmatrix} 0 \odot c \odot 0 & 0 \odot a + c \odot b \\ 0 \odot 0 + 0 \odot 0 & 0 \odot a + 0 \odot b \end{bmatrix} = \begin{bmatrix} 0 & \max\{0, c + b - 1\} \\ 0 & 0 \end{bmatrix}$$

The computation of the π closure yields

$$A_{\pi^*} = \begin{bmatrix} 0 & a \\ 0 & b \end{bmatrix}^* = \begin{bmatrix} f^* & \max\{0, \max\{0, f^* + a - 1\} + b^* - 1\} \\ (b \odot 0) \odot f^* & \max\{f^*, \dots\} \end{bmatrix} = \begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix}.$$

Satisfaction. Finally, let us define the (graded) satisfaction relation. A similar satisfaction relation was already considered, in the context of many-valued modal logics by M. Fitting in [Fit91, Fit92b] and, more recently, by F. Bou in [BEGR11].

As mentioned above, the carrier of \mathbf{A} corresponds to the space of truth degrees for $\mathcal{GDL}(\mathbf{A})$. Hence, the graded satisfaction relation for a model $\mathcal{A} \in \text{Mod}^{\mathcal{GDL}(\mathbf{A})}(\Delta)$, with \mathbf{A} complete, consists of a function

$$\vDash : W \times \text{Fm}^{\mathcal{GDL}(\mathbf{A})}(\Delta) \rightarrow \mathbf{A}$$

recursively defined as follows:

- $(w \vDash \top) = \top$
- $(w \vDash \perp) = \perp$
- $(w \vDash p) = V(p, w)$, for any $p \in \text{Prop}$
- $(w \vDash \rho \wedge \rho') = (w \vDash \rho) \cdot (w \vDash \rho')$
- $(w \vDash \rho \vee \rho') = (w \vDash \rho) + (w \vDash \rho')$
- $(w \vDash \rho \rightarrow \rho') = (w \vDash \rho) \rightarrow (w \vDash \rho')$
- $(w \vDash \rho \leftrightarrow \rho') = (w \vDash \rho \rightarrow \rho'); (w \vDash \rho' \rightarrow \rho)$

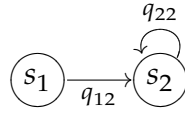
- $(w \vDash \langle \pi \rangle \rho) = \sum_{w' \in W} (A_{\pi}(w, w'); (w' \vDash \rho))$
- $(w \vDash [\pi] \rho) = \bigwedge_{w' \in W} (A_{\pi}(w, w') \rightarrow (w' \vDash \rho))$

We say that ρ is *valid* when, for any any model \mathcal{A} , and for each state $w \in W$, $(w \vDash \rho) = \top$. As stated above, we use, in the semantics of $\rho \leftrightarrow \rho$, the adjunct of the implication \rightarrow , instead of (standard) conjunction \cdot . Actually, the operation \rightarrow has a double role, acting as composition when modelling the space of computations, and as “strong conjunction” when referring to the truth space. We present below a simple illustration of our framework.

Example 2.3.2. *Let us fix the complete action lattice \mathbf{L} of Example 2.2.4 and a signature $(\{\pi\}, \{p\})$. Moreover, consider the model $\mathcal{A} = (W, V, (A_{\pi_0})_{\pi_0 \in \text{Prog}_0})$, with $W = \{s_1, s_2\}$ and the atomic program*

$$A_{\pi} = \begin{bmatrix} \perp & q_{12} \\ \perp & q_{22} \end{bmatrix}$$

which can be represented by the following labelled transition system:



In order to illustrate the method, let us consider the logic $\mathcal{GDL}()$. Assuming $V(s_1, p) = 0$ and $V(s_2, p) = 1$ we evaluate the very simple sentence $\langle \pi^* \rangle p$ in state s_1 . For this we calculate

$$\begin{aligned}
 (s_1 \vDash \langle \pi^* \rangle p) &= \sum_{w' \in W} (A_{\pi^*}(s_1, w'); (w' \vDash p)) \\
 &= \max \{ \\
 &\quad \max\{0, (s_1 \vDash p) + A_{\pi^*}(s_1, s_1) - 1\}, \\
 &\quad \max\{0, (s_2 \vDash p) + A_{\pi^*}(s_1, s_2) - 1\} \\
 &\quad \} \\
 &= \max\{0, q_{12}\} \\
 &= q_{12}
 \end{aligned}$$

This means that, we can assure, with a degree of certainty q_{12} , that p is achieved from s_1 through π^* .

2.4 Weighted sets and weighted relations

Although Kleene algebra and residuated lattices, or their combinations, represent well established mathematical abstractions for programs and propositions, there is a lack of suitable models to interpret them in the weighted case. The construction of such models is based on the theory of fuzzy sets and fuzzy relations [Zad65, Gog67], to embed the weights into the meaning of both programs and propositions.

Definition 2.4.1. *Let W be a set and \mathbf{A} a complete residuated lattice. A weighted subset of W is a function $\varphi : W \rightarrow \mathbf{L}$.*

The value $\varphi(w)$ defines the *membership degree of w in φ* . The set of all weighted subsets of W is denoted as \mathbf{L}^W .

Example 2.4.1. *Take the action lattice of example 2.2.4. Consider the colour “brown” which is composed by the three main components “red” (R), “green” (G) and “blue” (B). Such colour may be seen as a weighted set over $W = \{R, G, B\}$, i.e. a function φ defined as $\varphi(R) = 0,65$, $\varphi(G) = 0,16$ and $\varphi(B) = 0,16$, where the values 0,65, 0,16, and 0,16 are the respective colour proportions in a 0-1 scale.*

Note that the definition presented is neither the original, from [Zad65], where the unit interval $[0, 1]$ was taken as the set of truth values for weighted sets, nor its generalisation [Gog67] to \mathbf{L} -valued weighted sets, where \mathbf{L} can be a partial ordered set, a lattice, or a residuated lattice. Due to our approach in **Part 1** we adopt, instead, an extension of the latter over an arbitrary complete action lattice, by defining the operator $*$ as shown in the examples of Section 2.2.

Definition 2.4.2. *Let W_1, W_2, \dots, W_n be sets. A weighted relation μ between W_1, W_2, \dots, W_n is a weighted subset of the Cartesian product $W_1 \times W_2 \times \dots \times W_n$.*

For each $w_1 \in W_1, w_2 \in W_2, \dots, w_n \in W_n$, $\mu(w_1, w_2, \dots, w_n)$ can be interpreted as the truth value of how elements w_1, w_2, \dots, w_n are related by μ . Therefore, as weighted sets model collections of objects, weighted relations model relationships between objects up to some membership degree. Since in this thesis we work only with binary weighted relations, the term *weighted relation* always refers to weighted subsets of the Cartesian product $W_1 \times W_2$. The set of all weighted relations over W is denoted as $\mathbf{A}^{W \times W}$.

Example 2.4.2. *Take, again, the same lattice from the previous example. Consider the set $W = \{\text{Lisbon}, \text{London}, \text{Rio de Janeiro}\}$. It is possible to define a weighted relation μ representing “how close” the cities are from each other as $\mu(\text{Lisbon}, \text{London}) = 0.8$, $\mu(\text{Lisbon}, \text{Rio de Janeiro}) = 0.3$, $\mu(\text{Rio de Janeiro}, \text{London}) = 0.1$.*

Definition 2.4.3. *Let Σ be an alphabet, \mathbf{A} a complete action lattice and consider Σ^* the set of words over Σ . A weighted language over Σ is a weighted subset of Σ^* , that is, a function $\lambda : \Sigma^* \rightarrow \mathbf{A}$.*

Part I

WEIGHTED *SINGLE-FLOW* COMPUTATIONS

Context

Kleene algebra is a pervasive structure in computer science, applications ranging from semantics and logics of programs, to automata and formal language theory, as well as to the design and analysis of algorithms. Some recent examples show the applicability of Kleene algebra also to the analysis of hybrid systems [HM09], separation logic [DHM11] and non-termination analysis [DMS11]. As shown in 2, the axiomatisation of Kleene algebra forms a deductive system to manipulate programs [Koz94a]. Its applications typically deal with conventional, imperative programming constructs, namely conditionals and loops. Reasoning equationally about them entails the need for a notion of a test, which lead to the development of *Kleene algebra with tests* (KAT) [Koz97] combining the expressiveness of Kleene algebra with a Boolean subalgebra to formalise tests.

D. Kozen [Koz94a] proved that plain Kleene algebra is closed under the formation of box matrices, later extending this result to Kleene algebra with tests by seeing a test as a Boolean diagonal matrix. Instances of KAT include, but are not limited to, binary relations, which model programs as input-output relations between states, and guarded languages, as a formal semantics for guarded automata.

Hoare logic was the first formal system proposed for verification of programs. Introduced in 1969, its wide influence made it a cornerstone in program correctness. Hoare Logic encompasses a syntax to reason about Partial Correctness Assertions (PCA) of the form $\{\rho_1\}\pi\{\rho_2\}$, called Hoare triples, and a deductive system to reason about them [Hoa69, Flo93]. In a Hoare triple, ρ_1 and ρ_2 stand for predicates, representing the *pre* and *post* conditions, respectively, and π is a program statement. *Propositional Hoare logic* (PHL) is a fragment of Hoare Logic, in which Hoare triples are reduced to static assertions about the underlying domain of computation [Koz00].

Both a mathematical structure, providing an algebraic abstraction of programs behaviour, and a logic, to reason about their properties, are essential ingredients for a rigorous software engineering development methodologies. A well known way to combine those two components is *dynamic logic* [HKT00], whose development along the past twenty years went hand-in-hand with the evolution of its object, i.e. *the very notion of a program*. The result was the emergence of a plethora of dynamic logics tailored to specific programming paradigms. This ranges from the well-known classical case [FL79] to less conventional examples for which e.g. programs are compositions of actions in UML state machines [KMRG15] or event/actions regular expressions [HMK19]. Different rephrasings of what should count for a program in each specific context lead to different variants of dynamic logics: examples include probabilistic [Koz85], fuzzy [LY02], concurrent [Pel87], quantum [BS12] and continuous [Pla10] computations, and combinations thereof. The fragment introduced in reference [FL77], PDL, abstracts programs and formulas into a language built from propositions.

It was later proved that both the syntax and the deductive system of PHL is subsumed by PDL [HKT00] and KAT [Koz00]. The former translates a Hoare triple $\{\rho_1\}\pi\{\rho_2\}$ into the PDL formula $\rho_1 \rightarrow [\pi]\rho_2$, while the latter maps Hoare triples to equations and the rules of inference into implications between equations.

The characterisation of relations that identify states with equivalent behaviours is crucial to support a set of software development practices, including reuse, refinement and minimization of programs and models. On the logical view, these relations usually enjoy an invariance property, i.e. they preserve the satisfaction of formulas.

Overview

As originally presented, KAT, dynamic logic and Hoare logic are suitable frameworks to reason about classic imperative programs. In fact, such programs are particularly “well tractable”: they represent a sequence of discrete steps, each of them modelled as an atomic transition in a standard automaton. Typically, assertions about those programs have an outcome in a Boolean truth space. Our goal in **Part 1** is to provide mathematical support to generalise algebras and logics based on KAT, Dynamic Logic and Hoare Logic to model programs and assertions in weighted contexts. The results achieved will provide the formal setting to reason about \mathcal{F}_1 -programs interpreted as weighted “single-flow” computations.

To follow this programme, we first need an algebra of programs. In order to obtain such a structure, we present two generalisations of KAT, *graded Kleene algebra with tests* (GKAT) and *idempotent graded Kleene algebra with tests* (I-GKAT), while comparing their axiomatisations. We also illustrate both structures with a set of relevant examples and discuss some properties. Analogously to [Koz00], we encode PHL in I-GKAT and its while-free fragment in GKAT. Moreover, we examine in this setting a classic result of denesting two nested while loops in a weighted scenario. To interpret programs and assertions in a weighted context, we resort to the theory of weighted sets and weighted relations, proving that those endowed with suitable operators are instances of both GKAT and I-GKAT. In other words, the set of instances presented, namely the algebra of weighted sets $\mathbf{WSET}(\mathbf{K}, \mathbf{T})$ and the algebra of weighted relations $\mathbf{WREL}(\mathbf{K}, \mathbf{T})$, over complete residuated lattices \mathbf{K} and \mathbf{T} , will be used to give semantics to \mathcal{F}_1 -programs.

The development of dynamic logics for \mathcal{F}_1 -programs is based on reference [MNM16]. This paper initiated a research agenda on the systematic development of propositional, multi-valued dynamic logics parametric on an action lattice, which defines both the computational paradigm where programs live, and the truth space where assertions take value. We extend this agenda to an equational scenario, taking computational states as valuations of variables over a given domain, and programs as their modifiers. The idea is to capture weighted “single-flow” computations, i.e. imperative programs interpreted over different notions of ‘weighted’ computation — the very notion of *weight* being brought to scene as a parameter, encoded, as in [MNM16], in a complete action lattice, for the generation of the corresponding dynamic logic. Depending on each complete action lattice chosen, such weights will be interpreted as e.g. the uncertainty associated to the effectiveness of a particular computation, or a measure of the resources consumed, such as the energy or the execution time.

A second contribution of **Part 1** is the study of bisimulation for the models introduced above, which is, in our approach, defined parametrically on a complete action lattice. Finally, bisimilarity is shown to entail modal equivalence for the corresponding dynamic logic.

Roadmap

Part 1 is organised as follows. Chapter 3 discusses the algebraic constructions to model \mathcal{F}_1 -programs. In particular, Section 3.1 recapitulates some fundamental concepts. Section 3.2 introduces *graded Kleene algebra with tests (GKAT)* as a generalisation of KAT, detailing its axiomatisation, a few examples and proofs of basic properties. It also presents a partial encoding of PHL in GKAT. Section 3.3 introduces *idempotent graded Kleene algebra with tests (I-GKAT)* as another generalisation of the standard KAT and a refinement of GKAT, offering a complete encoding of PHL. Section 3.4 presents the sets of all weighted sets, weighted relations, weighted languages and $n \times n$ matrices, with the appropriate operations, as examples of GKAT and I-GKAT. Section 3.5 puts those frameworks in action, by discussing some equational proofs for program equivalences in a weighted scenario.

Chapter 4 presents a semantics for \mathcal{F}_1 -programs to include program variables and assignments. Chapter 5 extends the method proposed in [MNM16] to incorporate \mathcal{F}_1 -programs. All constructions are illustrated in detail for three paradigmatic parameters: the classical *Boolean lattice*, the *Gödel algebra* to capture vagueness in computation, and the *tropical semiring* to reason about resource consumption. An axiomatic system for the generated logics is presented in Section 5.1. Finally, results on bisimilarity and invariance are discussed in Section 5.4.

ALGEBRAS OF WEIGHTED *SINGLE-FLOW* COMPUTATIONS

As stated in the introduction, the frameworks developed in this thesis to model and reason about weighted “single-flow” computations are built on top of the triple *algebra-semantics-logic*. This chapter focuses on the first component, by developing generalisations of *Kleene algebra with tests* [Koz97] to model programs and assertions in a weighted setting.

3.1 Preliminaries: Kleene algebra with tests and Hoare logic

First of all, let us recall from [Koz97, Koz00] the basic concepts of KAT and PHL, and the relation between them.

Definition 3.1.1. *A Kleene algebra with tests (KAT) is a tuple*

$$(K, T, +, ;, *, \bar{}, 0, 1)$$

where $T \subseteq K$, 0 and 1 are constants in T , $+$ and $;$ are binary operators in both K and T , $*$ is a unary operator in K , and $\bar{}$ is a unary operator defined only on T such that:

- $(K, +, ;, *, 0, 1)$ is a Kleene algebra;
- $(T, +, ;, \bar{}, 0, 1)$ is a Boolean algebra;
- $(T, +, ;, 0, 1)$ is a subalgebra of $(K, +, ;, 0, 1)$.

The elements of K , denoted by lower case letters p, q, r, s, x, y, z , stand for programs and the elements of T , denoted by a, b, c, d are called tests. Operators “+” and “;” play a different role when acting on programs or tests. The former stands for non-deterministic choice over programs, and a form of logical disjunction on tests. The latter is taken as the sequential composition of actions when applied to elements of K , and as a “multiplication” of tests when applied to elements of T . Finally, in the domain of programs, constants 0 and 1 interpret the *halt* and *skip* commands. When applied to tests, they stand for the logical constants *false* and *true*, respectively. Some operators are specific to only tests or programs. For instance, operator $*$ stands for iterative execution of programs and operator $\bar{}$ corresponds to the negation \neg of a test. Kleene algebra with tests induces an abstract programming language, where conditionals and **while** loops programming constructs are encoded as follows:

$$\begin{aligned}
& \mathbf{if } b \mathbf{ then } p \stackrel{\text{def}}{=} b; p + \bar{b} \\
& \mathbf{if } b \mathbf{ then } p \mathbf{ else } q \stackrel{\text{def}}{=} b; p + \bar{b}; q \\
& \mathbf{while } b \mathbf{ do } p \stackrel{\text{def}}{=} (b; p)^*; \bar{b}
\end{aligned}$$

The encoding of Propositional Hoare Logic (PHL) in KAT leads to an equational calculus to reason about Hoare triples. As originally introduced [Hoa69], one such triple $\{b\}p\{c\}$ is valid if whenever precondition b is met, the postcondition c is guaranteed to hold, upon the successful termination of program p . Classically, validity in PHL is established through the set of rules in Figure 9.

- *Composition rule:*

$$\frac{\{b\}p\{c\} \quad \{c\}q\{d\}}{\{b\}p; q\{d\}}$$

- *Conditional rule:*

$$\frac{\{b \wedge c\}p\{d\}, \{\neg b \wedge c\}q\{d\}}{\{c\} \mathbf{if } b \mathbf{ then } p \mathbf{ else } q \{d\}}$$

- *While rule:*

$$\frac{\{b \wedge c\}p\{c\}}{\{c\} \mathbf{while } b \mathbf{ do } p\{\neg b \wedge c\}}$$

- *Weakening and Strengthening rule:*

$$\frac{b' \rightarrow b, \{b\}p\{c\}, c \rightarrow c'}{\{b'\} p\{c'\}}$$

Figure 9: Hoare logic rules.

A Hoare triple $\{b\}p\{c\}$ is encoded in KAT as $b; p; \bar{c} = 0$, which is equivalent to $b; p = b; p; c$. The first equation means, intuitively, that the execution of p with precondition b and postcondition \bar{c} does not halt. Equation $b; p = b; p; c$, on the other hand, states that the verification of the post condition c after the execution of $b; p$ is redundant. PHL inference rules are encoded in KAT, as follows:

- *Composition rule:*

$$b; p = b; p; c \quad \wedge \quad c; q = c; q; d \quad \Rightarrow \quad b; p; q = b; p; q; d$$

- *Conditional rule:*

$$b; c; p = b; c; p; d \quad \wedge \quad \bar{b}; c; q = \bar{b}; c; q; d \quad \Rightarrow \quad c; (b; p + \bar{b}; q) = c; (b; p + \bar{b}; q); d$$

- *While rule:*

$$b; c; p = b; c; p; c \Rightarrow c; (b; p)^*; \bar{b} = c; (b; p)^*; \bar{b}; \bar{b}; c$$

- *Weakening and Strengthening rule:*

$$b' \leq b \wedge b; p = b; p; c \wedge c \leq c' \Rightarrow b'; p = b'; p; c'$$

where \leq is a partial order on K defined by $p \leq q$ iff $p + q = q$.

3.2 Generalising Kleene algebra with tests: a first approach

The approach proposed in this thesis to reason about computations in a weighted context proceeds by redefining not only the notion of a program execution, but also the interpretation of assertions about programs. Since such assertions take the form of tests, we start by modifying the part of the axiomatisation of KAT that deals with properties of tests, i.e. the Boolean algebra $(T, +, \cdot, \bar{\cdot}, 0, 1)$.

Graded Kleene algebra with tests

Instead of having a Boolean outcome, as in KAT, tests are weighted, taking values from a truth space with more than two possible outcomes, a weighted truth space. As a consequence, the expression $b; p$ represents a weighted execution of program p , guarded by the value of test b . This leads to the following generalisation of KAT:

Definition 3.2.1. *A graded Kleene algebra with tests (GKAT) is a tuple*

$$(K, T, +, ;, *, \rightarrow, 0, 1)$$

where K and T are sets, with $T \subseteq K$, 0 and 1 are constants in T , $+$ and $;$ are binary operators in both K and T , $*$ is a unary operator in K , and \rightarrow is an operator only defined in T , satisfying axioms (25)-(38), where $p, q, r \in K$ and $a, b \in T$. Relation \leq is induced by $+$ in the usual way: $p \leq q$ iff $p + q = q$.

As in KAT, programs are elements of K denoted by lower case letters p, q, r, s, x, y, z and tests are elements of T denoted by a, b, c, d . Observe that a Kleene algebra is recovered by restricting the definition of GKAT to $(K, T, +, ;, *, 0, 1)$, axiomatised by (25)-(35). Operators $+$, $;$ and constants $0, 1$ in GKAT are interpreted as for KAT. The operator \rightarrow plays in GKAT the role of logical implication over tests. Note also that $(T, +, ;, 0, 1)$ is a subalgebra of $(K, +, ;, 0, 1)$. Differently from what happens in KAT, negation \bar{a} , for $a \in T$, is not explicitly denoted, although it can be derived as $a \rightarrow 0$.

$$p + (q + r) = (p + q) + r \quad (25) \qquad 1 + p; p^* = p^* \quad (32)$$

$$p + q = q + p \quad (26) \qquad 1 + p^*; p = p^* \quad (33)$$

$$p; (q; r) = (p; q); r \quad (27) \qquad q + p; r \leq r \Rightarrow p^*; q \leq r \quad (34)$$

$$p; 1 = 1; p = p \quad (28) \qquad q + r; p \leq r \Rightarrow q; p^* \leq r \quad (35)$$

$$p; (q + r) = (p; q) + (p; r) \quad (29) \qquad a; b \leq c \Leftrightarrow b \leq a \rightarrow c \quad (36)$$

$$(p + q); r = (p; r) + (q; r) \quad (30) \qquad a \leq 1 \quad (37)$$

$$p; 0 = 0; p = 0 \quad (31) \qquad a; b = b; a \quad (38)$$

Note that a GKAT can be characterised by two more equations, which we removed from Definition 3.1.1:

$$p + p = p \quad (39)$$

$$p + 0 = p \quad (40)$$

The two of them, however, can be derived from the axiomatisation above.

Lemma 3.2.1. *Equations (39) and (40) hold in any GKAT.*

Proof. For (39) we have, by (37),

$$\begin{aligned} & 1 + 1 = 1 \\ \Rightarrow & \quad \{ \text{monotonicity of } ; \} \\ & p; (1 + 1) = p; 1 \\ \Leftrightarrow & \quad \{ (29) \text{ and } (28) \} \\ & p + p = p \end{aligned}$$

We prove (40) by stating, by (37),

$$\begin{aligned} & 0 + 1 = 1 \\ \Rightarrow & \quad \{ \text{monotonicity of } ; \} \\ & p; (0 + 1) = p; 1 \\ \Leftrightarrow & \quad \{ (29) \} \\ & p; 0 + p; 1 = p; 1 \\ \Leftrightarrow & \quad \{ (28) \text{ and } (31) \} \\ & 0 + p = p \\ \Leftrightarrow & \quad \{ (26) \} \\ & p + 0 = p \end{aligned}$$

□

A main particularity of the GKAT axiomatization concerns rules (37) and (38), which form a weakened version of the axiomatization of a Boolean algebra. Indeed, GKAT generalises KAT in the following sense:

Lemma 3.2.2. *Any KAT is a GKAT.*

Proof. For a fixed KAT

$$A = (K, T, +, ;, *, \bar{}, 0, 1)$$

define

$$M = (K, T, +, ;, *, \rightarrow, 0, 1)$$

inheriting operators $+$, $;$, $*$ and constants 0 and 1 from A . Let $a \rightarrow b := \bar{a} + b$, for $a, b \in T$.

The crucial part of the proof verifies that axiom (36) holds for M , for all $a, b, c \in T$. To see that, assume $a; b \leq c$. Then,

$$\begin{aligned} & a; b \leq c \\ \Leftrightarrow & \quad \{ ; \text{ is the conjunction of tests} \} \\ & a \wedge b \leq c \\ \Leftrightarrow & \quad \{ \text{commutativity of } \wedge \} \\ & b \wedge a \leq c \\ \Leftrightarrow & \quad \{ \text{test shunting} \} \\ & b \leq \bar{a} + c \\ \Leftrightarrow & \quad \{ \text{definition of } \rightarrow \} \\ & b \leq a \rightarrow c \end{aligned}$$

Since axioms (25)-(35), (37) and (38) are axioms of A , M is indeed a GKAT. □

GKAT is able to model weighted computations and propositions evaluated in a weighted truth space. Instances of this structure range from a discrete scale to the real interval $[0, 1]$, as shown below. As stated in Lemma 3.2.2, some instances of GKAT can also be retrieved from the classical case. Such is the case of the Boolean lattice:

Example 3.2.1. *Our first example is the well-known Boolean lattice*

$$\mathbf{2} = (\{\top, \perp\}, \{\top, \perp\}, \vee, \wedge, *, \rightarrow, \perp, \top)$$

with the standard interpretation of Boolean connectives. Operator $*$ maps each element of $\{\top, \perp\}$ to \top and \rightarrow corresponds to logical implication.

Example 3.2.2. The second example is provided by the three-element linear lattice, which introduces an explicit denotation u for “unknown” (or “undefined”).

$$\mathbf{3} = (\{\top, u, \perp\}, \{\top, u, \perp\}, \vee, \wedge, *, \rightarrow, \perp, \top)$$

where

\vee	\perp	u	\top	\wedge	\perp	u	\top	\rightarrow	\perp	u	\top	$*$	\perp	\top
\perp	\perp	u	\top	\perp	\perp	\perp	\perp	\perp	\top	\top	\top	\perp	\top	\top
u	u	u	\top	u	\perp	u	u	u	\perp	\top	\top	u	\top	\top
\top	\top	\top	\top	\top	\perp	u	\top	\top	\perp	u	\top	\top	\top	\top

Example 3.2.3. For a fixed, finite set S , another instance of GKAT is

$$2^S = (P(S), P(S), \cup, \cap, *, \rightarrow, \emptyset, S)$$

where $P(S)$ denotes the powerset of S , \cup and \cap are set union and intersection, respectively, $*$ maps each set $X \in P(S)$ into S , and $X \rightarrow Y = X^C \cup Y$, where $X^C = \{x \in S \mid x \notin X\}$.

Example 3.2.4. As another example, consider the standard \mathbf{P} algebra

$$\mathbf{P} = ([0, 1], [0, 1], \max, \cdot, *, \rightarrow, 0, 1)$$

where \cdot is the usual multiplication of real numbers,

$$x \rightarrow y = \begin{cases} 1, & \text{if } x \leq y \\ y/x, & \text{if } y < x \end{cases}$$

$/$ is the division of reals and $*$ maps each point of the interval $[0, 1]$ to 1.

Example 3.2.5. A Gödel algebra is also an instance of GKAT. Actually,

$$\mathbf{G} = ([0, 1], [0, 1], \max, \min, *, \rightarrow, 0, 1)$$

where

$$x \rightarrow y = \begin{cases} 1, & \text{if } x \leq y \\ y, & \text{if } y < x \end{cases}$$

Example 3.2.6. Another example is the well-known Łukasiewicz arithmetic lattice.

$$\mathbf{L} = ([0, 1], [0, 1], \max, \ominus, *, \rightarrow, 0, 1)$$

where $x \rightarrow y = \min\{1, 1 - x + y\}$, $x \odot y = \max\{0, x + y - 1\}$ and $*$ maps each point of the interval $[0, 1]$ to 1.

Example 3.2.7. Let us consider now a GKAT endowing the finite Wajsberg hoop with a star operator [BFOO]. For a fixed natural k and a generator a , one gets

$$W_k = (W_k, W_k, +, ;, *, \rightarrow, 0, 1)$$

where $W_k = \{a^0, a^1, \dots, a^{k-1}\}$, $1 = a^0$ and $0 = a^{k-1}$. Moreover, for any $m, n \leq k - 1$, $a^m + a^n = a^{\min\{m,n\}}$, $a^m ; a^n = a^{\min\{m+n, k-1\}}$, $(a^m)^* = a^0$ and $a^m \rightarrow a^n = a^{\max\{n-m, 0\}}$.

Example 3.2.8. The $(\min, +)$ Kleene algebra [Koz92], known as the tropical semiring, can be extended to a GKAT by adding residuation \rightarrow . First, let R_+ denote the set $\{x \in \mathbb{R} \mid x \geq 0\}$ and adjoin $+\infty$ as a new constant. Thus, define

$$\mathbf{R} = (R_+ \cup \{+\infty\}, R_+ \cup \{+\infty\}, \min, +_{\mathbf{R}}, *, \rightarrow, +\infty, 0_{\mathbf{R}})$$

where, for any $x, y \in R_+ \cup \{+\infty\}$, $x^* = 0_{\mathbf{R}}$ and $x \rightarrow y = \max\{y - x, 0\}$.

By abuse of notation we use the same notation for the complete action lattices presented in Section 2.2 (Examples (2.2.1)-(2.2.11)) and the instances of GKAT listed above. Examples 3.2.2 and 3.2.7 represent instances to reason in discrete multi-valued logics. Examples 3.2.4, 3.2.5 and 3.2.6, in their turn, are particularly relevant to model fuzzy and continuous multi-valued logics.

Modelling weighted assertions where the outcome is not Boolean entails the need for weakening the subalgebra $(T, +, ;, *, \rightarrow, 0, 1)$ of KAT. As a consequence, it is not necessarily true that $a + (a \rightarrow 0) = 1$, as it happens in Boolean algebras. Let us illustrate this in the following example.

Example 3.2.9. Consider the GKAT

$$(\{0, n, m, 1\}, \{0, m, 1\}, +, ;, *, \rightarrow, 0, 1)$$

in which the operation $*$ maps all points to the top element 1, and the remaining operations are defined as follows:

$+$	0	n	m	1	$;$	0	n	m	1	\rightarrow	0	n	m	1
0	0	n	m	1	0	0	0	0	0	0	1	0	1	1
n	n	n	m	1	n	0	0	0	n	n	0	0	0	0
m	m	m	m	1	m	0	0	0	m	m	m	0	1	1
1	1	1	1	1	1	0	n	m	1	1	0	0	m	1

Clearly, $a = m$ entails $m + (m \rightarrow 0) = m + m = m \neq 1$.

The example above illustrates that tests can assume a wider range of values in GKAT, representing the truth degree of the statement “ b holds”. The expression $b;p$ means that the execution of a program p is guarded by that particular value.

Encoding propositional Hoare logic in GKAT

Section 3.1 discusses framing PHL (syntax and inference rules) in equations and quasi-equations in KAT. Similarly, exploring a possible encoding of propositional Hoare logic into GKAT provides a smooth way for reasoning about the correctness of programs with some form of weight associated to their execution. Since this new structure deals with graded tests, both the meaning of Hoare triples and the inference rules need to be adjusted. This reinterpretation leads to a generalised version we shall refer to as *graded propositional Hoare logic (GPHL)*.

In the presence of graded tests, the interpretation of a triple $\{b\}p\{c\}$, and hence, the correctness of a program, relies on the idea that whenever $b;p$ executes with truth degree b , if and when it halts, it is guaranteed that $(b;p);c$ holds with at least the same truth degree. In other words, the computation of the correctness degree is monotonic for sequential composition. Therefore, the encoding in GKAT is captured by the following inequality:

$$b;p \leq b;p;c$$

Note that the equivalence

$$b;p \leq b;p;c \Leftrightarrow b;p = b;p;c, \quad (41)$$

holds in GKAT, following directly from (29), (37) and (28).

The inference rules of Hoare logic are encoded in GKAT as follows.

Theorem 3.2.1. *The following implications are theorems in GKAT.*

1. *Composition rule:*

$$b;p \leq b;p;c \wedge c;q \leq c;q;d \Rightarrow b;p;q = b;p;q;d$$

2. *Conditional rule:*

$$b;c;p \leq b;c;p;d \wedge (b \rightarrow 0);c;q \leq (b \rightarrow 0);c;q;d \Rightarrow \\ c;(b;p + (b \rightarrow 0);q) \leq c;(b;p + (b \rightarrow 0);q);d$$

3. *Weakening and Strengthening rule:*

$$b' \leq b \wedge b; p \leq b; p; c \wedge c \leq c' \Rightarrow b'; p \leq b'; p; c'$$

Proof.

1. COMPOSITION RULE: Let us assume that $b; p \leq b; p; c$ and $c; q \leq c; q; d$. By (41), these inequalities are equivalent to $b; p = b; p; c$ and $c; q = c; q; d$, respectively. So, we have

$$\begin{aligned} & b; p; q \\ = & \{ b; p = b; p; c \} \\ & b; p; c; q \\ = & \{ c; q = c; q; d \} \\ & b; p; c; q; d \\ = & \{ b; p = b; p; c \} \\ & b; p; q; d \end{aligned}$$

2. CONDITIONAL RULE: Assume $b; c; p \leq b; c; p; d$ and $(b \rightarrow 0); c; q \leq (b \rightarrow 0); c; q; d$. First of all, observe that, for any $p, q, r, s \in K$

$$p \leq q \wedge r \leq s \Rightarrow p + r \leq q + s \quad (42)$$

Because $p \leq q$ and $r \leq s$, i.e. $p + q = q$ and $r + s = s$, then, by (25) and (26), $(p + r) + (q + s) = (p + q) + (r + s) = q + s$. So, by (42),

$$\begin{aligned} & b; c; p + (b \rightarrow 0); c; q \leq b; c; p; d + (b \rightarrow 0); c; q; d. \\ \Leftrightarrow & \{ (38), (29) \text{ and } (30) \} \\ & c; (b; p + (b \rightarrow 0); q) \leq c; (b; p + (b \rightarrow 0); q); d \end{aligned}$$

3. WEAKENING AND STRENGTHENING RULE: Observe that, for all $b, c \in T$ and $p \in K$,

$$b; p \leq b; p; c \Rightarrow b; p; (c \rightarrow 0) \leq 0 \quad (43)$$

Using (41) to rewrite (43) as

$$b; p = b; p; c \Rightarrow b; p; (c \rightarrow 0) = 0 \quad (44)$$

and, assuming $b;p = b;p;c$, we have

$$\begin{aligned}
& b;p;(c \rightarrow 0) \\
= & \{ b;p = b;p;c \text{ assumption} \} \\
& b;p;c;(c \rightarrow 0) \\
= & \{ a;(a \rightarrow 0) = 0 \text{ and (31)} \} \\
& 0
\end{aligned}$$

Using (44), the Weakening and Strengthening rule can be rewritten as

$$a \leq b \wedge b;p;(c \rightarrow 0) = 0 \wedge (d \rightarrow 0) \leq (c \rightarrow 0) \Rightarrow a;p;(d \rightarrow 0) = 0$$

which follows from the monotonicity of “;”.

□

The attentive reader certainly noticed the absence of an encoding of the While rule in the graded setting. In analogy with what was done before, such a rule would take the form:

$$b;c;p \leq b;c;p;c \Rightarrow c;(b;p)^*;(b \rightarrow 0) \leq c;(b;p)^*;(b \rightarrow 0);(b \rightarrow 0);c \quad (45)$$

However, this is not necessarily true for all $p \in K$ and $b, c \in T$. To see this, consider the GKAT structure of Example 3.2.9.

If $b = 0, c = m, p = 0$, by (31) and (40), the instantiation of $b;c;p \leq b;c;p;c$ boils down to

$$0;m;0 + 0;m;0;m = 0;m;0;m \Leftrightarrow 0 = 0$$

and that of $c;(b;p)^*;(b \rightarrow 0) \leq c;(b;p)^*;(b \rightarrow 0);(b \rightarrow 0);c$ becomes, by (31), (28) and (40),

$$m;(0)^*;1 + m;(0)^*;1;1;m = m;(0)^*;1;1;m \Leftrightarrow m = 0$$

Using these two equations, implication (45) boils down to $0 = 0 \Rightarrow m = 0$, which is obviously false.

3.3 Generalising Kleene algebra with tests: an idempotent variant

By carefully observing the encoding of the PHL while rule in KAT, it becomes apparent that one cause of failure of an analogous encoding in GKAT, mentioned in the previous section, is the impossibility of duplicating graded tests. Actually, in GKAT, $b;b = b$ does not hold, but only the weaker form $b;b \leq b$. The solution proposed is to refine the GKAT structure with some additional properties such that, i) it allows for a complete encoding of Hoare logic and, at the same time, ii) captures non-Boolean examples, assigning truth degrees to program execution and evaluation of tests. The idea is still to resort to a weaker algebra to model the tests, instead of the Boolean algebra implicitly used in KAT.

Definition 3.3.1. An idempotent graded Kleene algebra with tests (I-GKAT) is a tuple

$$(K, T, +, ;, *, \rightarrow, 0, 1)$$

where K and T are sets, with $T \subseteq K$, 0 and 1 are constants in T , $+$ and $;$ are binary operations in both K and T , $*$ is a unary operator in K , and \rightarrow is an operator only defined in T , satisfying axioms (25)-(38) plus the axiom below:

$$a; a = a \tag{46}$$

Note that, as in GKAT, negation is not explicitly denoted, but can be derived as $a \rightarrow 0$.

The following result establishes I-GKAT as a strict subclass of GKAT, as well as another generalisation of KAT. Examples 3.2.1, 3.2.2, 3.2.3, 3.2.5 and the two-element 3.2.7 are instances of I-GKAT. Figure 10 sums up our results.

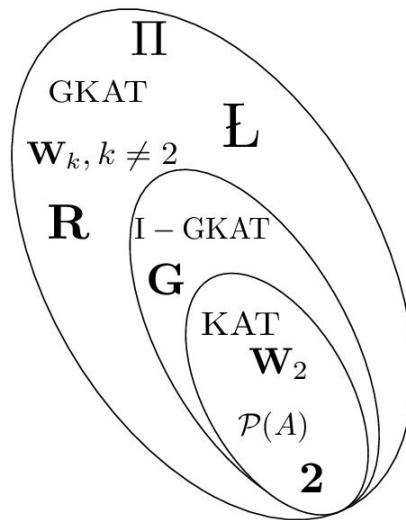


Figure 10: Examples of KAT, GKAT and I-GKAT.

Lemma 3.3.1. *Any KAT is a I-GKAT, which in turn is also a GKAT.*

Proof. It suffices to show that axiom (36) holds for all $a, b, c \in T$. The proof is similar to that of Lemma 3.2.2. \square

Encoding propositional Hoare logic in I-GKAT

After refining the basic structure, the remaining of this section discusses how to encode propositional Hoare logic in I-GKAT. Differently from what happens in GKAT, the three encodings proposed by D. Kozen for Hoare logic are equivalent in I-GKAT:

$$b;p = b;p;c \Leftrightarrow b;p \leq b;p;c \Leftrightarrow b;p \leq p;c$$

Hence, the inference rules of Hoare logic can be encoded in I-GKAT as they are in classical propositional Hoare logic.

Theorem 3.3.1. *The following implication is a theorem in I-GKAT.*

$$b;c;p \leq b;c;p;c \Rightarrow c;(b;p)^*; (b \rightarrow 0) \leq c;(b;p)^*; (b \rightarrow 0); (b \rightarrow 0); c$$

Proof. Assume, by (38),

$$b;c;p \leq b;c;p;c \Leftrightarrow c;b;p \leq c;b;p;c \tag{47}$$

Let us start by proving

$$\begin{aligned} & c + c;(b;p)^*;c;b;p \\ \leq & \quad \{ \text{by (71)} \} \\ & c + c;(b;p)^*;c;b;p;c \\ \leq & \quad \{ \text{by (46) and (28)} \} \\ & c;1;c + c;(b;p)^*;c;b;p;c \\ \leq & \quad \{ \text{by distributivity} \} \\ & c;(1 + (b;p)^*;c;b;p);c \\ \leq & \quad \{ \text{by monotonicity} \} \\ & c;(1 + (b;p)^*;b;p);c \\ \leq & \quad \{ \text{by (14)} \} \\ & c;(b;p)^*;c \end{aligned}$$

On the other hand,

$$\begin{aligned}
& c + c; (b;p)^*; c; b;p \leq c; (b;p)^*; c \\
\Rightarrow & \quad \{ (35) \} \\
& c; (b;p)^* \leq c; (b;p)^*; c \\
\Rightarrow & \quad \{ \text{monotonicity of } ; \} \\
& c; (b;p)^*; (b \rightarrow 0) \leq c; (b;p)^*; c; (b \rightarrow 0) \\
\Leftrightarrow & \quad \{ (38) \} \\
& c; (b;p)^*; (b \rightarrow 0) \leq c; (b;p)^*; (b \rightarrow 0); c \\
\Leftrightarrow & \quad \{ (46) \} \\
& c; (b;p)^*; (b \rightarrow 0) \leq c; (b;p)^*; (b \rightarrow 0); (b \rightarrow 0); c
\end{aligned}$$

□

3.4 Weighted structures and matrices as GKAT/I-GKAT

Binary relations model classic imperative programs as input/output pairs of states, constituting one of the standard models of Kleene algebra with tests [EC96]. However, the behaviour of weighted computations calls for other formalisms.

Since they were introduced as concepts based on natural language and fuzzy logic, weighted sets and weighted relations [AZ96] are natural candidates to model weighted computations. In fact, weighted sets describe the membership degree of elements in a collection. In its turn, weighted relations contain input-output pairs of states with a value describing how strong the relation is.

This section illustrates both GKAT and I-GKAT constructions by discussing how they can be developed over weighted sets, weighted relations, weighted languages and square matrices.

Consider two complete residuated lattices \mathbf{K} and \mathbf{T} over, respectively, carriers K and T . Weighted sets, weighted relations and weighted languages may be presented as functions from their domain to, respectively, K and T . We denote by $+$ the supremum of \mathbf{K} , and operators $;$ and \rightarrow satisfying axioms (25)-(31) and (36). We use the same notation for operators of \mathbf{T} satisfying (25)-(31) plus (36)-(38). Since $+$ and $;$ are associative and have identity, we can generalise them to n -ary operators. We use the notation \sum and \prod to represent their iterated versions, respectively. For the specific constructions presented in this section (definitions 3.4.1, 3.4.2 and 3.4.3), we assume both \mathbf{K} and \mathbf{T} to be complete residuated lattices, ensuring that the following properties hold:

$$a; \left(\sum_{i \in I} b_i \right) = \sum_{i \in I} (a; b_i) \quad (48)$$

$$\left(\sum_{i \in I} b_i\right); a = \sum_{i \in I} (b_i; a) \quad (49)$$

where I is a (possibly infinite) index set. To formalise these structures as I-GKAT, we consider $;$ to be also idempotent, i.e. to satisfy (46).

Definition 3.4.1. *Let X be a set and \mathbf{T} a complete residuated lattice. The algebra of weighted sets over \mathbf{T} is the structure*

$$\mathbf{WSET}(\mathbf{T}) = (\mathbf{T}^X, \mathbf{T}^X, \cup, \otimes, *, \rightarrow, \emptyset, \chi)$$

where \mathbf{T}^X is the set of all weighted sets over X and, for all $\varphi, \psi \in \mathbf{T}^X$ and $x \in X$, operators are defined pointwise by

$$\begin{aligned} (\varphi \cup \psi)(x) &= \varphi(x) + \psi(x) \\ (\varphi \otimes \psi)(x) &= \varphi(x); \psi(x) \\ (\varphi^*)(x) &= \sum_{k \geq 0} \varphi^k(x) \\ (\varphi \rightarrow \psi)(x) &= \varphi(x) \rightarrow \psi(x) \\ \emptyset(x) &= \mathbf{0} \\ \chi(x) &= \mathbf{1} \end{aligned}$$

with $\varphi^0(x) = \chi(x)$ and $\varphi^{k+1}(x) = (\varphi^k \otimes \varphi)(x)$. The values of weighted sets, $\varphi(x)$ and $\psi(x)$, are elements of \mathbf{T} , and $\mathbf{0}, \mathbf{1}$ are constants. The partial order \subseteq for weighted sets is given by

$$\varphi \subseteq \psi \Leftrightarrow \forall_{x \in X} \cdot \varphi(x) \leq \psi(x), \varphi, \psi \in \mathbf{T}^X$$

where \leq is the order of Definition 2.2.1.

Note that, in this definition, the two sets of the signature of $\mathbf{WSET}(\mathbf{T})$ coincide, both defined as functions with codomain \mathbf{T} .

Theorem 3.4.1. *For any complete residuated lattice \mathbf{T} satisfying (38), $\mathbf{WSET}(\mathbf{T})$ forms a GKAT and. If \mathbf{T} satisfies (46), $\mathbf{WSET}(\mathbf{T})$ forms a I-GKAT.*

Proof. Considering the way that elements of \mathbf{T}^X and operators \cup, \otimes and \rightarrow are defined, it is straightforward to verify that axioms (25) to (31) and (36) to (38), plus (46) for I-GKAT, are satisfied. We prove that axioms dealing with operator $*$ ((32)- (34)) hold as well. Axiom (35) can be proved analogously to (34).

Axiom (32):

$$(\chi \cup (\varphi \otimes \varphi^*))(x)$$

$$\begin{aligned}
&= \quad \{ \text{definition of } \mathbf{T}^X \} \\
&\quad \chi(x) + \varphi(x); \varphi^*(x) \\
&= \quad \{ \text{definition of } \varphi^*(x) \} \\
&\quad \chi(x) + \varphi(x); \left(\sum_{k \geq 0} \varphi^k(x) \right) \\
&= \quad \{ \text{definition of } \sum \} \\
&\quad \chi(x) + \varphi(x); (\varphi^0(x) + \varphi^1(x) + \dots) \\
&= \quad \{ (48) \} \\
&\quad \chi(x) + \varphi(x); \varphi^0(x) + \varphi(x); \varphi^1(x) + \dots \\
&= \quad \{ \text{definition of } \varphi^{k+1}(x) \} \\
&\quad \chi(x) + \varphi(x) + \varphi^2(x) + \dots \\
&= \quad \{ \text{definition of } \sum \} \\
&\quad \varphi^*(x)
\end{aligned}$$

The proof of (33) is analogous but using (49).

Axiom (34)

Let us assume $(\varphi \otimes \psi)(x) \leq \psi(x)$, i.e. $\varphi(x); \psi(x) \leq \psi(x)$, by definition of the operators on weighted sets. Moreover,

$$\begin{aligned}
&(\varphi^* \otimes \psi)(x) \\
&= \quad \{ \text{definitions of } * \text{ and } \otimes \} \\
&\quad \left(\sum_{k \geq 0} \varphi^k(x) \right); \psi(x) \\
&= \quad \{ \text{definition of } \sum \} \\
&\quad (\varphi^0(x) + \varphi^1(x) + \dots); \psi(x) \\
&= \quad \{ (49) \text{ and } (28) \} \\
&\quad \psi(x) + \varphi(x); \psi(x) + \dots
\end{aligned}$$

By hypothesis and given that $\varphi(x); \varphi(x) \leq \varphi(x)$, for all $\varphi(x) \in T$, we conclude that

$$\psi(x) + \varphi(x); \psi(x) + \dots \leq \psi(x)$$

□

Definition 3.4.2. *Let X be a set, \mathbf{K} and \mathbf{T} complete residuated lattices. The algebra of weighted relations over \mathbf{K} and \mathbf{T} is defined as*

$$\mathbf{WREL}(\mathbf{K}, \mathbf{T}) = (\mathbf{K}^{X \times X}, \mathbf{T}^{X \times X}, \cup, \circ, *, \rightarrow, \emptyset, D)$$

where $\mathbf{K}^{X \times X}$ is the set of all weighted relations over $X \times X$, the elements of $\mathbf{T}^{X \times X}$ are diagonal weighted relations, i.e. weighted relations σ such that $\sigma(x, y) = \mathbf{0}$ whenever $x \neq y$. Moreover, for all $\mu, \nu \in \mathbf{K}^{X \times X}$, $\sigma, \eta \in \mathbf{T}^{X \times X}$, $x, y, z \in X$, the operators are defined by

$$\begin{aligned} (\mu \cup \nu)(x, y) &= \mu(x, y) + \nu(x, y) \\ (\mu \circ \nu)(x, y) &= \sum_{z \in X} \mu(x, z); \nu(z, y) \\ (\mu^*)(x, y) &= \sum_{k \geq 0} \mu^k(x, y) \\ (\sigma \rightarrow \eta)(x, y) &= \begin{cases} \sigma(x, y) \rightarrow \eta(x, y), & \text{if } x = y \\ \mathbf{0}, & \text{otherwise} \end{cases} \\ \emptyset(x, y) &= \mathbf{0} \\ D(x, y) &= \begin{cases} \mathbf{1}, & \text{if } x = y \\ \mathbf{0}, & \text{otherwise} \end{cases} \end{aligned}$$

with $\mu^0(x, y) = D(x, y)$, $\mu^{k+1}(x, y) = (\mu^k \circ \mu)(x, y)$. The values of weighted relations, $\mu(x, y)$ and $\nu(x, y)$, are elements of \mathbf{K} , the values of $\sigma(x, y)$ and $\eta(x, y)$ are elements of \mathbf{T} , and, finally, $\mathbf{0}, \mathbf{1}$ are constants. Similarly to the previous one, the partial order \subseteq for weighted relations is given by

$$\mu \subseteq \nu \Leftrightarrow \forall_{(x,y) \in X \times X} \cdot \mu(x, y) \leq \nu(x, y), \mu, \nu \in \mathbf{K}^X$$

where \leq is the order referred in Definition 2.2.1.

Theorem 3.4.2. Given complete residuated lattices \mathbf{K} and \mathbf{T} , with \mathbf{T} satisfying (38), $\mathbf{WREL}(\mathbf{K}, \mathbf{T})$ is a GKAT. If \mathbf{T} satisfies (46), $\mathbf{WREL}(\mathbf{K}, \mathbf{T})$ is also a l-GKAT.

Proof. The validity of (25) and (26) follows immediately from the definitions of operators on weighted relations. Let $\mu, \nu, \xi \in \mathbf{K}^{X \times X}$ and $x, y, z, w \in X$.

Axiom (27):

$$\begin{aligned} &((\mu \circ \nu) \circ \xi)(x, y) \\ = &\quad \{ \text{definition of } \circ \} \\ &\sum_{z \in X} \left(\sum_{w \in X} (\mu(x, w); \nu(w, z)); \xi(z, y) \right) \\ = &\quad \{ \text{definition of } \sum \text{ and } z_i, w_i \in X, 1 \leq i \leq n \} \end{aligned}$$

$$\begin{aligned}
& (\mu(x, w_1); \nu(w_1, z_1) + \cdots + \mu(x, w_n); \nu(w_n, z_1)); \tilde{\xi}(z_1, y) + \cdots \\
& + (\mu(x, w_1); \nu(w_1, z_n) + \cdots + \mu(x, w_n); \nu(w_n, z_n)); \tilde{\xi}(z_n, y) \\
= & \quad \{ (49) \text{ and } (27) \} \\
& \mu(x, w_1); (\nu(w_1, z_1); \tilde{\xi}(z_1, y)) + \cdots + \mu(x, w_n); (\nu(w_n, z_1); \tilde{\xi}(z_1, y)) + \cdots \\
& + \mu(x, w_1); (\nu(w_1, z_n); \tilde{\xi}(z_n, y)) + \cdots + \mu(x, w_n); (\nu(w_n, z_n); \tilde{\xi}(z_n, y)) \\
= & \quad \{ (26) \text{ and } (48) \} \\
& \mu(x, w_1); (\nu(w_1, z_1); \tilde{\xi}(z_1, y) + \cdots + \nu(w_1, z_n); \tilde{\xi}(z_n, y)) + \cdots \\
& + \mu(x, w_n); (\nu(w_n, z_1); \tilde{\xi}(z_1, y) + \cdots + \nu(w_n, z_n); \tilde{\xi}(z_n, y)) \\
= & \quad \{ \text{definition of } \sum \} \\
& \sum_{w \in X} (\mu(x, w); (\sum_{z \in X} (\nu(w, z); \tilde{\xi}(z, y)))) \\
= & \quad \{ \text{definition of } \circ \} \\
& (\mu \circ (\nu \circ \tilde{\xi}))(x, y)
\end{aligned}$$

Axiom (28):

$$\begin{aligned}
& (\mu \circ D)(x, y) \\
= & \quad \{ \text{definition of } \circ \} \\
& \sum_{z \in X} \mu(x, z); D(z, y) \\
= & \quad \{ \text{definition of } \sum \text{ and } z_i \in X, 1 \leq i \leq n \} \\
& \mu(x, z_1); D(z_1, y) + \cdots + \mu(x, z_n); D(z_n, y) \\
= & \quad \{ \text{definition of } D \} \\
& \mu(x, z_1); 1 + \cdots + \mu(x, z_n); 1, \text{ for all } D(z_i, y) = 1, 1 \leq i \leq n \\
= & \quad \{ (28) \} \\
& \mu(x, z_1) + \cdots + \mu(x, z_n) \\
= & \quad \{ \text{definition of } \mu \} \\
& \mu(x, y)
\end{aligned}$$

Axiom (29):

$$\begin{aligned}
& (\mu \circ (\nu \cup \zeta))(x, y) \\
= & \quad \{ \text{definitions of } \circ \text{ and } \cup \} \\
& \sum_{z \in X} \mu(x, z); (\nu(z, y) + \zeta(z, y)) \\
= & \quad \{ \text{definition of } \sum \text{ and } z_i \in X, 1 \leq i \leq n \} \\
& \mu(x, z_1); (\nu(z_1, y) + \zeta(z_1, y)) + \cdots + \mu(x, z_n); (\nu(z_n, y) + \zeta(z_n, y)) \\
= & \quad \{ (29) \} \\
& \mu(x, z_1); \nu(z_1, y) + \mu(x, z_1); \zeta(z_1, y) + \cdots + \mu(x, z_n); \nu(z_n, y) + \mu(x, z_n); \zeta(z_n, y) \\
= & \quad \{ (26) \} \\
& \mu(x, z_1); \nu(z_1, y) + \cdots + \mu(x, z_n); \nu(z_n, y) \\
& \quad + \mu(x, z_1); \zeta(z_1, y) + \cdots + \mu(x, z_n); \zeta(z_n, y) \\
= & \quad \{ \text{definition of } \sum \} \\
& \sum_{z \in X} \mu(x, z); \nu(z, y) + \sum_{z \in X} \mu(x, z); \zeta(z, y) \\
= & \quad \{ \text{definitions of } \circ \text{ and } \cup \text{ on weighted relations} \} \\
& ((\mu \circ \nu) \cup (\mu \circ \zeta))(x, y)
\end{aligned}$$

Axioms (30) to (35): The proof of Axiom (30) is analogous. Axiom (31) follows straightforwardly, since $\emptyset(x, y) = 0$ is the absorbing element of $;$ over \mathbf{K} . Axioms (32)-(35) are proved as in Theorem 3.4.1, but taking instead the composition of weighted relations, i.e.

$$(\mu \circ \nu)(x, y) = \sum_{z \in X} \mu(x, z); \nu(z, y)$$

for all $\mu, \nu \in \mathbf{K}^{X \times X}$. As in Theorem 3.4.1, we only verify axioms (32) and (34). The validity of (33) and (35) is left for the reader, since the arguments used are essentially the same of (34).

Axiom (32):

$$\begin{aligned}
& (D \cup (\mu \circ \mu^*))(x, y) \\
= & \quad \{ \text{definition of } \cup, \circ \text{ and } * \} \\
& D(x, y) + \sum_{z \in X} (\mu(x, z); (\sum_{k \geq 0} \mu^k(z, y))) \\
= & \quad \{ \text{definition of } \mu \} \\
& D(x, y) + \sum_{z \in X} (\mu(x, z); \mu^0(z, y) + \mu(x, z); \mu(z, y) + \dots) \\
= & \quad \{ \text{definition of } \sum \text{ and } z_i \in X, 1 \leq i \leq n \} \\
& D(x, y) + \mu(x, z_1); \mu^0(z_1, y) + \mu(x, z_1); \mu(z_1, y) + \dots \\
& \quad + \mu(x, z_n); \mu^0(z_n, y) + \mu(x, z_n); \mu(z_n, y) + \dots \\
= & \quad \{ \text{definition of } \mu^k \} \\
& D(x, y) + \mu(x, y) + \mu(x, y) + \dots + \mu(x, y) + \mu(x, y) + \dots \\
= & \quad \{ (26) \text{ and } (39) \} \\
& D(x, y) + \mu(x, y) + \mu(x, y) + \dots \\
= & \quad \{ \text{definition of } \mu^k \} \\
& \mu^*(x, y)
\end{aligned}$$

Axiom (34): We assume the left side of the implication of (34) for elements of K , i.e.

$$(\mu \circ \nu)(x, y) \leq \nu(x, y) \Leftrightarrow \sum_{z \in X} \mu(x, z); \nu(z, y) \leq \nu(x, y)$$

by definition of \circ on weighted relations.

$$\begin{aligned}
& (\mu^* \circ \nu)(x, y) \\
= & \quad \{ \text{definitions of } \circ \text{ and } * \} \\
& \sum_{z \in X} \left(\left(\sum_{k \geq 0} \mu^k(x, z) \right); \nu(z, y) \right) \\
= & \quad \{ \text{definition of } \sum, (49) \text{ and } z_i \in X, 1 \leq i \leq n \} \\
& \mu^0(x, z_1); \nu(z_1, y) + \mu(x, z_1); \nu(z_1, y) + \dots \\
& + \mu^0(x, z_n); \nu(z_n, y) + \mu(x, z_n); \nu(z_n, y) + \dots
\end{aligned} \tag{50}$$

Resorting to (26) and the hypothesis, the terms of (50) are re-organised as follows. For $k = 0$

$$\mu^0(x, z_1); \nu(z_1, y) + \dots + \mu^0(x, z_n); \nu(z_n, y) \leq \nu(x, y)$$

and for $k = 1$

$$\mu(x, z_1); \nu(z_1, y) + \cdots + \mu(x, z_n); \nu(z_n, y) \leq \nu(x, y).$$

Each term $\mu^k(x, z_i); \nu(z_i, y)$, for $k \geq 2$, for each z_i , $1 \leq i \leq n$, becomes

$$(\mu \circ \cdots \circ \mu)(x, z_i); \nu(z_i, y) = \sum_{w^1 \in X} (\cdots \sum_{w^k \in X} (\mu(x, w^k); \mu(w^k, w^{k-1})); \cdots; \mu(w^1, z_i)); \nu(z_i, y)$$

Using (49), (27) and the hypothesis, we can simplify the expression and prove (34). As an example, the term for $k = 2$, for each z_i is computed as follows.

$$\begin{aligned} & \mu^2(x, z_i); \nu(z_i, y) \\ = & \quad \{ \text{definition of } \mu^i \text{ of Definition 3.4.2} \} \\ & (\mu \circ \mu)(x, z_1); \nu(z_i, y) \\ = & \quad \{ \text{definition of } \circ \} \\ & (\sum_{w \in X} (\mu(x, w); \mu(w, z_i))); \nu(z_i, y) \\ = & \quad \{ \text{definition of } \sum \text{ and } w_i \in X, 1 \leq i \leq n \} \\ & (\mu(x, w_1); \mu(w_1, z_i) + \cdots + \mu(x, w_n); \mu(w_n, z_i)); \nu(z_i, y) \\ = & \quad \{ (49) \text{ and } (27) \} \\ & \mu(x, w_1); (\mu(w_1, z_i); \nu(z_i, y)) + \cdots + \mu(x, w_n); (\mu(w_n, z_i); \nu(z_i, y)) \\ \leq & \quad \{ \mu(x, z); \nu(z, y) \leq \nu(z, y) \text{ for all } x, y, z \in X \text{ and monotonicity of } ; \text{ and } + \} \\ & \mu(x, w_1); \nu(w_1, y) + \cdots + \mu(x, w_n); \nu(w_n, y) \\ = & \quad \{ \text{hypothesis} \} \\ & \sum_{w \in X} \mu(x, w); \nu(w, y) \leq \nu(x, y) \end{aligned}$$

$1 \leq i \leq k$. Generalisation to other values for k is straightforward.

So, we prove that (50) becomes $\nu(x, y) + \cdots + \nu(x, y)$, reduced by (39) to $\nu(x, y)$.

Axiom (36) (" \Rightarrow "): Let $\sigma, \eta, \theta \in \mathbf{T}^{X \times X}$ and assume

$$\begin{aligned} & (\sigma \circ \eta)(x, y) \leq \theta(x, y) \\ \Leftrightarrow & \quad \{ \text{definition of } \circ \} \\ & \sum_{z \in X} \sigma(x, z); \eta(z, y) \leq \theta(x, y) \\ \Leftrightarrow & \quad \{ \text{definition of } \sum \text{ and } z_i \in X, 1 \leq i \leq n \} \\ & \sigma(x, z_1); \eta(z_1, y) + \cdots + \sigma(x, z_n); \eta(z_n, y) \leq \theta(x, y) \end{aligned}$$

Since $\sigma(x, z_i), \eta(z_i, y) \in \mathbf{T}^{X \times X}$, there is, at most, one $1 \leq i \leq n$ such that $x = z_i$ and $z_i = y$. So, $\sigma(x, z_1); \eta(z_1, y) + \cdots + \sigma(x, z_n); \eta(z_n, y) = \sigma(x, z_i); \eta(z_i, y) \leq \theta(x, y)$, for the only $1 \leq i \leq n$ such that $x = z_i$ and $z_i = y$. Since $\sigma(x, z_i), \eta(z_i, y)$ and $\theta(x, y) \in \mathbf{T}$, by (36) on \mathbf{T} , $\sigma(x, z_i); \eta(z_i, y) \leq \theta(x, y)$ implies $\eta(x, y) \leq \sigma(x, y) \rightarrow \theta(x, y)$. The proof of (" \Leftarrow ") is analogous.

Axiom (37): The proof of (37) is trivial, since $\sigma(x, y) \leq 1 = \Delta(x, y)$, for all $\sigma(x, y) \in \mathbf{T}^{X \times X}$.

Axiom (38): First observe that

$$\begin{aligned}
& (\sigma \circ \eta)(x, y) \\
= & \quad \{ \text{definition of } \circ \} \\
& \sum_{z \in X} \sigma(x, z); \eta(z, y) \\
= & \quad \{ \text{definition of } \sum \text{ and } z_i \in X, 1 \leq i \leq n \} \\
& \sigma(x, z_1); \eta(z_1, y) + \cdots + \sigma(x, z_n); \eta(z_n, y), \\
& \text{for all } \sigma(x, z_i), \eta(z_i, y) \neq 0, \text{ with } 1 \leq i \leq n
\end{aligned}$$

Clearly $x = z_i = y$, using the definition of $\sigma(x, y)$, for all $\sigma \in \mathbf{T}^{X \times X}$. Thus, the proof follows directly from (38) on elements of T , as shown below.

$$\begin{aligned}
& \eta(x, z_1); \sigma(z_1, y) + \cdots + \eta(x, z_n); \sigma(z_n, y) \\
= & \quad \{ \text{definition of } \sum \} \\
& \sum_{z \in X} \eta(x, z); \sigma(z, y) \\
= & \quad \{ \text{definition of } \circ \} \\
& (\eta \circ \sigma)(x, y)
\end{aligned}$$

To prove that **WREL**(\mathbf{T}) is also a I-GKAT, for any complete residuated lattice \mathbf{T} , we need to prove axiom (46).

Axiom (46):

$$\begin{aligned}
& (\sigma \circ \sigma)(x, y) \\
= & \quad \{ \text{definition of } \circ \} \\
& \sum_{z \in X} \sigma(x, y); \sigma(z, y) \\
= & \quad \{ \text{definition of } \sum \text{ and } z_i \in X, 1 \leq i \leq n \} \\
& \sigma(x, z_1); \sigma(z_1, y) + \cdots + \sigma(x, z_n); \sigma(z_n, y)
\end{aligned}$$

Again, $\sigma(x, y) + \sigma(x, z_1); \eta(z_1, y) + \dots + \sigma(x, z_n); \eta(z_n, y)$ reduces to $\sigma(x, z_i); \sigma(z_i, y)$, for the only $1 \leq i \leq n$ such that $x = z_i = y$. But $\sigma(x, z_i); \sigma(z_i, y) = \sigma(x, y)$, by (46), since $\sigma(x, z_i), \sigma(z_i, y) \in \mathbf{T}$. \square

Definition 3.4.3. Let Σ be an alphabet, Σ^* the set of all words over Σ and \mathbf{K}, \mathbf{T} complete residuated lattices. The algebra of weighted languages over \mathbf{K}, \mathbf{T} is defined as

$$\mathbf{WLANG}(\mathbf{K}, \mathbf{T}) = (K^{\Sigma^*}, T^{\Sigma^*}, \cup, \cdot, *, \rightarrow, \emptyset, \epsilon)$$

where \mathbf{K}^{Σ^*} stands for the set of all weighted languages over Σ . The elements of \mathbf{T}^{Σ^*} are languages defined, for each $t \in T$, by

$$t_t(a_1 \dots a_n) = \begin{cases} t, & \text{if } a_1 \dots a_n = \epsilon, \text{ with } \epsilon \text{ being the empty word} \\ \mathbf{0}, & \text{otherwise} \end{cases}$$

and, for all $\lambda_1, \lambda_2 \in K^{\Sigma^*}$ and all $\iota_1, \iota_2 \in T^{\Sigma^*}$, given a word $a_1 \dots a_n \in \Sigma^*$, the operators $\cup, \cdot, *, \rightarrow, \emptyset$ and ϵ are defined as:

$$\begin{aligned} (\lambda_1 \cup \lambda_2)(a_1 \dots a_n) &= \lambda_1(a_1 \dots a_n) + \lambda_2(a_1 \dots a_n) \\ (\lambda_1 \cdot \lambda_2)(a_1 \dots a_n) &= \sum_{i=1}^{n-1} \lambda_1(a_1 \dots a_i); \lambda_2(a_{i+1} \dots a_n) \\ (\lambda^*)(a_1 \dots a_n) &= \sum_{k \geq 0} \lambda^k(a_1 \dots a_n) \\ (\iota_1 \rightarrow \iota_2)(a_1 \dots a_n) &= \begin{cases} \prod_{a_1 \dots a_{i-1}} (\iota_1(a_1 \dots a_{i-1}) \rightarrow \iota_2(a_1 \dots a_n)), i \leq n, \\ \text{if } a_1 \dots a_{i-1} = \epsilon \\ \mathbf{0}, & \text{otherwise} \end{cases} \\ \emptyset(a_1 \dots a_n) &= \mathbf{0} \\ \epsilon(a_1 \dots a_n) &= \begin{cases} \mathbf{1} & \text{if } a_1 \dots a_n = \epsilon, \text{ with } \epsilon \text{ being the empty word} \\ \mathbf{0} & \text{otherwise} \end{cases} \end{aligned}$$

with $\lambda^0(a_1 \dots a_n) = \epsilon(a_1 \dots a_n)$ and $\lambda^{k+1}(a_1 \dots a_n) = (\lambda^k \cdot \lambda)(a_1 \dots a_n)$. The values of weighted sets, $\lambda_1(a_1 \dots a_n)$ and $\lambda_2(a_1 \dots a_n)$, are elements of \mathbf{K} , and $\mathbf{0}, \mathbf{1}$ are constants. The partial order \subseteq for weighted languages is given by

$$\lambda_1 \subseteq \lambda_2 \Leftrightarrow \forall_{a_1 \dots a_n \in \Sigma^*} \cdot \lambda_1(a_1 \dots a_n) \leq \lambda_2(a_1 \dots a_n), \lambda_1, \lambda_2 \in \mathbf{K}^{\Sigma^*}$$

where \leq is the order of Definition 2.2.1.

Theorem 3.4.3. Given complete residuated lattices \mathbf{K} and \mathbf{T} , with \mathbf{T} satisfying (38) and (46), $\mathbf{WLANG}(\mathbf{K}, \mathbf{T})$ is a l-GKAT.

Proof. Since a weighted language λ is a weighted subset of a set of elements (in this case, the alphabet Σ^*) and the operators \cup and $*$ are defined as \cup and $*$ in **WSET**(\mathbf{T}), respectively, and \cdot as \circ in **WREL**(\mathbf{K}, \mathbf{T}), the proof is identical to that of Theorem 3.4.1 for the cases of \cup and $*$, and to that of Theorem 3.4.2 for the \cdot operator.

It remains to prove axiom (36): Take $l_1, l_2, l_3 \in \mathbf{T}^{\Sigma^*}$ and $v \in \Sigma^*$. Consider first the case $v \neq \epsilon$. Assuming

$$(l_1 \cdot l_2)(v) \leq l_3(v) \Leftrightarrow \sum_{v_1, v_2} l_1(v_1); l_2(v_2) \leq l_3(v) \Leftrightarrow l_3(v) = 0$$

we want to prove that $l_2(v) \leq (l_1 \rightarrow l_3)(v)$. But, by definition of ι and \rightarrow ,

$$l_2(v) \leq (l_1 \rightarrow l_3)(v) \Leftrightarrow 0 \leq 0$$

Consider now $v = \epsilon$. We want to prove that

$$(l_1 \cdot l_2)(\epsilon) \leq l_3(\epsilon) \Leftrightarrow l_2(\epsilon) \leq (l_1 \rightarrow l_3)(\epsilon)$$

$$\begin{aligned} & l_2(\epsilon) \leq (l_1 \rightarrow l_3)(\epsilon) \\ \Leftrightarrow & \quad \{ \text{definition of } \rightarrow \} \\ & l_2(\epsilon) \leq \prod_u (l_1(u) \rightarrow l_3(u\epsilon)) \\ \Leftrightarrow & \quad \{ \text{definition of } \prod \} \\ & l_2(\epsilon) \leq (l_1(u_1) \rightarrow l_3(u_1\epsilon)); \dots; (l_1(u_{n-1}) \rightarrow l_3(u_{n-1}\epsilon)); (l_1(\epsilon) \rightarrow l_3(\epsilon\epsilon)), \\ & \quad u_1, \dots, u_{n-1} \neq \epsilon \\ \Leftrightarrow & \quad \{ \text{definition of } \iota \} \\ & l_2(\epsilon) \leq (0 \rightarrow 0); \dots; (0 \rightarrow 0); (l_1(\epsilon) \rightarrow l_3(\epsilon)) \\ \Leftrightarrow & \quad \{ 0 \rightarrow 0 = 1 \text{ for all } \mathbb{I}\text{-action lattices ([MNM16]) and (28)} \} \\ & l_2(\epsilon) \leq l_1(\epsilon) \rightarrow l_3(\epsilon) \\ \Leftrightarrow & \quad \{ (36) \} \\ & l_1(\epsilon); l_2(\epsilon) \leq l_3(\epsilon) \\ \Leftrightarrow & \quad \{ \text{definition of } \cdot \} \\ & (l_1 \cdot l_2)(\epsilon) \leq l_3(\epsilon) \end{aligned}$$

□

Now we present the construction of square matrices over a GKAT and I-GKAT.

Definition 3.4.4. Let $A = (\mathbf{K}, \mathbf{T}, +, ;, *, \rightarrow, 0, 1)$ be a GKAT (or a I-GKAT). Define the following structure over the family of $n \times n$ matrices.

$$\mathbb{M}_n(A) = (M_n(\mathbf{K}), \Delta_n(\mathbf{T}), +, ;, *, \rightarrow, 0_n, I_n)$$

where $+$ and $;$ stand for the usual matrix addition and multiplication, respectively; 0_n is the $n \times n$ matrix of zeros and I_n the $n \times n$ identity matrix. The subalgebra over the set $\Delta_n(\mathbf{T})$ of $n \times n$ diagonal matrices keep operators $+$ and $;$ and matrices 0_n and I_n defined as before. The entries of the diagonal matrices are elements of the subalgebra $(\mathbf{T}, +, ;, 0, 1)$ of GKAT (or I-GKAT) Finally, operation \rightarrow is defined as:

$$A \rightarrow B = \begin{cases} A_{ij} \rightarrow B_{ij} & \text{if } i = j \\ \mathbf{0} & \text{otherwise} \end{cases}$$

Theorem 3.4.4. $M_n(A)$ is a GKAT (and a I-GKAT).

Proof. It was already proved by Kozen [Koz94a] that the structure

$$(M_n(\mathbf{K}), +, ;, *, 0_n, I_n)$$

forms a Kleene algebra. Then, it remains to prove that

$$(\Delta_n(\mathbf{T}), +, ;, \rightarrow, 0_n, I_n)$$

satisfies axioms (36)-(38).

$$\text{Let } A = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix}, B = \begin{bmatrix} b_{11} & 0 & \cdots & 0 \\ 0 & b_{22} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & b_{nn} \end{bmatrix} \text{ and } C = \begin{bmatrix} c_{11} & 0 & \cdots & 0 \\ 0 & c_{22} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & c_{nn} \end{bmatrix}$$

be elements of $\Delta_n(T)$.

For (36) we prove that $A;B + C = C \Rightarrow B + A \rightarrow C = A \rightarrow C$. Using the definitions of the operators, we obtain

$$\begin{aligned} & \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix} ; \begin{bmatrix} b_{11} & 0 & \cdots & 0 \\ 0 & b_{22} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & b_{nn} \end{bmatrix} + \begin{bmatrix} c_{11} & 0 & \cdots & 0 \\ 0 & c_{22} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & c_{nn} \end{bmatrix} \\ &= \begin{bmatrix} c_{11} & 0 & \cdots & 0 \\ 0 & c_{22} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & c_{nn} \end{bmatrix} \end{aligned}$$

which is equivalent to

$$\begin{bmatrix} a_{11}; b_{11} + c_{11} & 0 & \cdots & 0 \\ 0 & a_{22}; b_{22} + c_{22} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & a_{nn}; b_{22} + c_{nn} \end{bmatrix} = \begin{bmatrix} c_{11} & 0 & \cdots & 0 \\ 0 & c_{22} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & c_{nn} \end{bmatrix}$$

In order for two matrices to be equal, their elements must be equal in the corresponding positions. So, the assumption is

$$\begin{cases} a_{11}; b_{11} + c_{11} = c_{11} \\ a_{22}; b_{22} + c_{22} = c_{22} \\ \cdots \\ a_{nn}; b_{nn} + c_{nn} = c_{nn} \end{cases}$$

which is verified as follows.

$$\begin{aligned} & \begin{bmatrix} b_{11} & 0 & \cdots & 0 \\ 0 & b_{22} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & b_{nn} \end{bmatrix} + \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix} \rightarrow \begin{bmatrix} c_{11} & 0 & \cdots & 0 \\ 0 & c_{22} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & c_{nn} \end{bmatrix} \\ & = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix} \rightarrow \begin{bmatrix} c_{11} & 0 & \cdots & 0 \\ 0 & c_{22} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & c_{nn} \end{bmatrix} \\ & \Leftrightarrow \begin{bmatrix} b_{11} + a_{11} \rightarrow c_{11} & 0 & \cdots & 0 \\ 0 & b_{22} + a_{22} \rightarrow c_{22} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & b_{22} + a_{22} \rightarrow c_{22} \end{bmatrix} \\ & = \begin{bmatrix} a_{11} \rightarrow c_{11} & 0 & \cdots & 0 \\ 0 & a_{22} \rightarrow c_{22} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & a_{nn} \rightarrow c_{nn} \end{bmatrix} \end{aligned}$$

i. e.

$$\begin{cases} b_{11} + a_{11} \rightarrow c_{11} = a_{11} \rightarrow c_{11} \\ b_{22} + a_{22} \rightarrow c_{22} = a_{22} \rightarrow c_{22} \\ \cdots \\ b_{nn} + a_{nn} \rightarrow c_{nn} = a_{nn} \rightarrow c_{nn} \end{cases}$$

But, since $a_{ij}, b_{ij}, c_{ij} \in T$ for all $1 \leq i, j \leq n$ it is verified by axiom (36) of GKAT that

$$\left\{ \begin{array}{l} a_{11}; b_{11} + c_{11} = c_{11} \Rightarrow b_{11} + a_{11} \rightarrow c_{11} = a_{11} \rightarrow c_{11} \\ a_{22}; b_{22} + c_{22} = c_{22} \Rightarrow b_{22} + a_{22} \rightarrow c_{22} = a_{22} \rightarrow c_{22} \\ \dots \\ a_{nn}; b_{nn} + c_{nn} = c_{nn} \Rightarrow b_{nn} + a_{nn} \rightarrow c_{nn} = a_{nn} \rightarrow c_{nn} \end{array} \right.$$

The proof for \Leftarrow is similar. The proofs of axioms (37) and (38) are analogous, using the definitions of the operators over elements of $\Delta_n(\mathbf{T})$. Note, in particular, the proof of axiom (38). It is well known that the multiplication of matrices is not commutative. However, since axiom (38) is only applied to elements of $\Delta_n(\mathbf{T})$, that is, diagonal matrices, and the multiplication of diagonal matrices is commutative, this axiom is valid in this context.

To prove that this also forms a l-GKAT, it suffices to show the validity of (46). The proof is similar to the one presented for GKAT, for all $A, B, C \in \Delta_n(\mathbf{T})$, using the definitions of the operators over elements of $\Delta_n(\mathbf{T})$. \square

3.5 An illustration: a folk theorem

To illustrate our framework we revisit, in this section, a previously mentioned result on denesting two nested **while** loops [Koz97], in a scenario where both assertions and programs are expressed in a weighted context. Most proofs in [Koz97] rely on the use of a commutativity condition ($b; p = p; b$) which asserts that the execution of program p does not modify the value of test b . In KAT, it is possible to argue, as well, that if p does not affect b , neither should it affect \bar{b} , which is formally stated through the following lemma:

Lemma 3.5.1. *In any Kleene algebra with tests the following are equivalent:*

$$(1) b; p = p; b$$

$$(2) \bar{b}; p = p; \bar{b}$$

$$(3) b; p; \bar{b} + \bar{b}; p; b = 0$$

In GKAT, negation is relaxed and expressed as $a \rightarrow 0$, for all $a \in T$. So, the conditions above must be written as

$$(1) b; p = p; b$$

$$(2) (b \rightarrow 0); p = p; (b \rightarrow 0)$$

$$(3) b; p; (b \rightarrow 0) + (b \rightarrow 0); p; b = 0$$

However, it is important to note that not all implications hold in GKAT.

Lemma 3.5.2. (1) \Leftrightarrow (2) does not hold in GKAT.

Proof. This can be shown by the following counter example: a GKAT over the set $\{0, n, m, 1\}$, with $\{0, n, 1\} \subseteq T$ and $m \in K$, in which the operator $*$ maps all points to the top element 1 and the remaining operators are defined as follows:

+	0	n	m	1	;	0	n	m	1	\rightarrow	0	n	m	1
0	0	n	m	1	0	0	0	0	0	0	1	0	<i>n.d.</i>	1
n	n	n	m	1	n	0	0	0	n	n	0	0	<i>n.d.</i>	1
m	m	m	m	1	m	0	n	m	m	m	<i>n.d.</i>	<i>n.d.</i>	<i>n.d.</i>	<i>n.d.</i>
1	1	1	1	1	1	0	n	m	1	1	0	0	<i>n.d.</i>	1

If $b = n, p = m$, the instantiation of $b; p; b \Leftrightarrow (b \rightarrow 0); p; (b \rightarrow 0)$ becomes

$$n; m = m; n \Leftrightarrow (n \rightarrow 0); m = m; (n \rightarrow 0)$$

Thus, the expression turns into $0 = n \Leftrightarrow 0 = 0$, which is clearly false. □

Lemma 3.5.3. Implications (1) \Rightarrow (3) and (2) \Rightarrow (3) hold in GKAT.

Proof. Both implications arise by commutativity and the fact that $a; (a \rightarrow 0) = 0$, for all $a \in T$. □

The intuitive interpretation of these implications corresponds to that if p preserves b (or $b \rightarrow 0$), the execution of p between testing b and its complement, no matter which test is performed first, results in halting.

Lemma 3.5.4. Implication (3) \Rightarrow (1) does not hold in GKAT.

Proof. This can be shown by the following counter example: a GKAT over the set $\{0, n, m, 1\}$, with $\{0, n, 1\} \subseteq T$ and $m \in K$, in which the operator $*$ maps all points to the top element 1 and the remaining operators are defined as follows:

+	0	n	m	1	;	0	n	m	1	\rightarrow	0	n	m	1
0	0	n	m	1	0	0	0	0	0	0	1	1	<i>n.d.</i>	1
n	n	n	m	1	n	0	0	n	n	n	n	1	<i>n.d.</i>	1
m	m	m	m	1	m	0	0	m	m	m	<i>n.d.</i>	<i>n.d.</i>	<i>n.d.</i>	<i>n.d.</i>
1	1	1	1	1	1	0	n	m	1	1	0	n	<i>n.d.</i>	1

If $b = n, p = m$, the instantiation of $b; p; (b \rightarrow 0) + (b \rightarrow 0); p; b = 0 \Rightarrow b; p = p; b$ becomes $0 = 0 \Rightarrow n = 0$ which is obviously false. □

Lemma 3.5.5. Implication (3) \Rightarrow (2) does not hold in GKAT.

Proof. Consequence of Lemma 3.5.2 and Lemma 3.5.3. □

These implications are interpreted as follows: if program p being executed between testing b and its complement $b \rightarrow 0$ (no matter which test is performed first) results in halting, then the execution of p preserves b (or $b \rightarrow 0$). A similar result holds for I-GKAT and is proved along similar lines.

We can therefore argue that this dependency on commutativity conditions becomes a hindrance for proving several, usual results on program equivalence: it is impossible to handle such proofs in a (quasi) equational way without considering them. However, the result that is, perhaps, the most interesting one, of denesting two nested **while** loops, does not resort to the commutativity conditions, and therefore can be shown to hold.

Its original proof [Koz97] relies on one of De Morgan laws:

$$\neg(a \vee b) \equiv \neg a \wedge \neg b$$

that can be formalised in our setting as

$$(a + b) \rightarrow 0 = (a \rightarrow 0); (b \rightarrow 0) \tag{51}$$

Since, in general, this rule does not hold in I-GKAT, we impose it to prove the result below. Note that the rule holds in all instances of I-GKAT enumerated in this chapter, namely (3.2.1), (3.2.2), (3.2.3), (3.2.5) and the two-element (3.2.7).

We are now in conditions to show that a pair of **while** loops can be transformed into a single **while** loop inside a conditional test, as formalised in the following theorem:

Theorem 3.5.1. *The program*

$$\begin{array}{l} \mathbf{while} \ b \ \mathbf{do} \ \mathbf{begin} \\ \quad p; \\ \quad \quad \mathbf{while} \ c \ \mathbf{do} \ q \\ \quad \mathbf{end} \\ \mathbf{end} \end{array} \tag{52}$$

is equivalent to

$$\begin{array}{l} \mathbf{if} \ b \ \mathbf{then} \ \mathbf{begin} \\ \quad p; \\ \quad \quad \mathbf{while} \ b + c \ \mathbf{do} \\ \quad \quad \quad \mathbf{if} \ c \ \mathbf{then} \ q \ \mathbf{else} \ p \\ \quad \quad \mathbf{end} \\ \mathbf{end} \end{array} \tag{53}$$

in I-GKAT extended with (51).

Proof. The proof uses an analogous reasoning of [Koz97]. First of all, we need the following identities:

$$p; (q; p)^* = (p; q)^*; p \tag{54}$$

$$p^*; (q; p^*)^* = (p + q)^* \quad (55)$$

which are derivable from the axioms of Kleene algebra and were proved in [Koz94a].

Let us now translate programs (52) and (53) to the language of I-GKAT. Program (52) becomes

$$(b; p; (c; q)^*; (c \rightarrow 0))^*; (b \rightarrow 0), \quad (56)$$

and (53) becomes¹

$$b; p; ((b + c); (c; q + (c \rightarrow 0); p))^*; ((b + c) \rightarrow 0) + (b \rightarrow 0) \quad (57)$$

Simplifying (61),

$$\begin{aligned} & (b; p; (c; q)^*; (c \rightarrow 0))^*; (b \rightarrow 0) \\ = & \quad \{ (32) \} \\ & (1 + b; p; (c; q)^*; (c \rightarrow 0); (b; p; (c; q)^*; (c \rightarrow 0))^*; (b \rightarrow 0) \\ = & \quad \{ (30) \} \\ & (b \rightarrow 0) + b; p; (c; q)^*; (c \rightarrow 0); (b; p; (c; q)^*; (c \rightarrow 0))^*; (b \rightarrow 0) \\ = & \quad \{ (54) \} \\ & (b \rightarrow 0) + b; p; (c; q)^*; ((c \rightarrow 0); b; p; (c; q)^*)^*; (c \rightarrow 0); (b \rightarrow 0) \end{aligned}$$

For (57), the sub expression $(b + c); (c; q + (c \rightarrow 0); p)$ becomes

$$\begin{aligned} & (b + c); (c; q + (c \rightarrow 0); p) \\ = & \quad \{ (29) \} \\ & b; c; q + b; (c \rightarrow 0); p + c; c; q + c; (c \rightarrow 0); p \\ = & \quad \{ (46), a; (a \rightarrow 0) = 0, (31) \text{ and } (40) \} \\ & b; c; q + b; (c \rightarrow 0); p + c; q \\ = & \quad \{ (39) \text{ and } (38) \} \\ & b; c; q + c; q + (c \rightarrow 0); b; p \\ = & \quad \{ (30) \} \\ & (b + 1); c; q + (c \rightarrow 0); b; p \end{aligned}$$

Moreover, $(b + c) \rightarrow 0 = (b \rightarrow 0); (c \rightarrow 0)$, by (51). Applying these transformations on (57), we obtain

¹ As in Kozen's paper [Koz97], we interpret the program **if** b **then** p as an abbreviation for a conditional test with the dummy **else** clause i.e., as the program $b; p + \bar{b} (b; p + b \rightarrow 0)$ in our setting).

$$b;p;(c;q + (c \rightarrow 0));b;p)^*; (b \rightarrow 0); (c \rightarrow 0) + (b \rightarrow 0)$$

Now, we need to prove that

$$\begin{aligned} & (b \rightarrow 0) + b;p;(c;q)^*; (c \rightarrow 0); (b;p;(c;q)^*)^*; (c \rightarrow 0); (b \rightarrow 0) \\ & = b;p;(c;q + (c \rightarrow 0));b;p)^*; (b \rightarrow 0); (c \rightarrow 0) + (b \rightarrow 0) \end{aligned}$$

But, by monotonicity of operators $+$ and $;$, such an expression is equivalent to

$$(c;q)^*; (c \rightarrow 0); (b;p;(c;q)^*)^* = (c;q + (c \rightarrow 0));b;p)^*$$

which is just an instance of the denesting rule (60). □

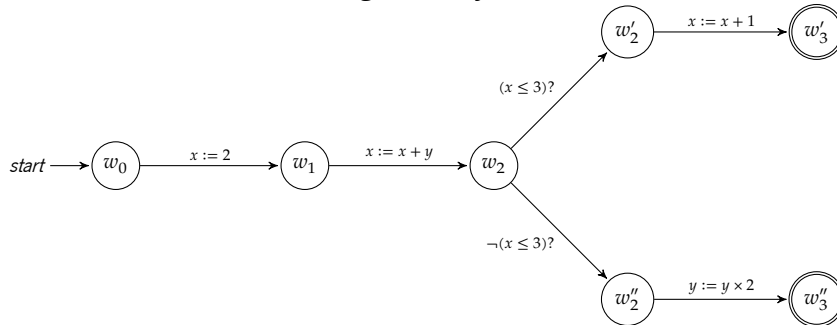
A WEIGHTED *SINGLE-FLOW* SEMANTICS

We now focus on the development of a denotational semantics for weighted “single-flow” computations. Along the chapter we will illustrate the introduced concepts by the program of Example 1.3.1, recalled below.

Example 4.0.1. Consider the imperative program

$$x := 2; x := x + y; (\text{if } x \leq 3 \text{ then } x := x + 1 \text{ else } y := y \times 2)$$

Its execution can be represented by the following transition system, where the conditional statement is encoded as a sum of alternatives guarded by a test.



The semantics of \mathcal{F}_1 -programs is defined over state spaces whose elements are weighted valuations of variables, i.e. functions $w : X \rightarrow \mathbf{A}^{\mathbb{R}}$, where \mathbf{A} is a complete action lattice. We denote the set of all states by $\mathbf{A}^{X \times \mathbb{R}}$.

An arbitrary \mathcal{F}_1 -program is generated as an expression described by the following rule

$$\pi ::= \text{skip} \mid \pi_0 \mid \rho? \mid \pi; \pi \mid \pi + \pi \mid \pi^* \quad (58)$$

with $\pi_0 \in \text{Prog}_0$, and $\rho?$ standing for a suitable notion of *test*. The latter, however, needs to be handled with some care: indeed the meaning of a test will depend on the generated logic and therefore on \mathbf{A} itself, as we will discuss in the next chapter when defining its semantics in terms of the satisfaction relation. For the moment, it is enough to notice that choice (+), iteration (*) and tests ($\rho?$) encode the usual constructs for conditionals and loops as considered in Table (3).

Definition 4.0.1 (Interpretation of functional terms). *Let F be a set of function symbols. The interpretation of a functional term $t \in T_F(X)$ in a state $w \in W$, is given by the map*

$$\llbracket _ \rrbracket_w : T_F(X) \rightarrow \mathbf{A}^{\mathbb{R}}$$

defined recursively as follows:

- $\llbracket x \rrbracket_w(r) = w(x)(r)$
- $\llbracket c \rrbracket_w(r) = \begin{cases} \top & \text{if } r = c \\ \perp & \text{otherwise} \end{cases}$
- $\llbracket f(t_1, \dots, t_n) \rrbracket_w(r) = \sum_{i \in I} \left\{ \prod_{j=1}^n \llbracket t_j \rrbracket_w(r_j^i) \mid f(r_1^i, \dots, r_n^i) = r \right\}$, where I is the cardinality of the set of all possible solutions of $f(r_1^i, \dots, r_n^i) = r$ in \mathbb{R} , with each f of arity n being interpreted as a function on real numbers $\mathbb{R}^n \rightarrow \mathbb{R}$ (e.g. $+$, \times , 2 , $\sqrt{_}$, ...), and c representing both the constant c and its syntactic representation.

Example 1.3.1 may help to illustrate this interpretation. Consider a set of states $W = \{w_0, w_1, w_2\}$, a set of variables $X = \{x, y\}$ and the complete action lattice in Example 2.2.7. Define state w_0 as follows.

$$w_0(x)(r) = \begin{cases} 0.5 & \text{if } r = 1 \\ 0.2 & \text{if } r = 2 \\ 0 & \text{otherwise} \end{cases} \quad w_0(y)(r) = \begin{cases} 1 & \text{if } r = 2 \\ 0 & \text{otherwise} \end{cases}$$

$$w_0(y)(r) = \begin{cases} 0.1 & \text{if } r = 1 \\ 0.4 & \text{if } r = 2 \\ 0 & \text{otherwise} \end{cases}$$

The interpretation of the term $x + y$ in w_0 is given by:

$$\begin{aligned} \llbracket x + y \rrbracket_{w_0}(2) &= \llbracket x \rrbracket_{w_0}(1); \llbracket y \rrbracket_{w_0}(1) = \min\{0.5, 0.1\} = 0.1 \\ \llbracket x + y \rrbracket_{w_0}(3) &= \llbracket x \rrbracket_{w_0}(1); \llbracket y \rrbracket_{w_0}(2) + \llbracket x \rrbracket_{w_0}(2); \llbracket y \rrbracket_{w_0}(1) \\ &= w_0(x, 1); w_0(y, 2) + w_0(x, 2); w_0(y, 1) \\ &= \max\{\min\{0.5; 0.4\}, \min\{0.2; 0.1\}\} = 0.4 \\ \llbracket x + y \rrbracket_{w_0}(4) &= \llbracket x \rrbracket_{w_0}(2); \llbracket y \rrbracket_{w_0}(2) = \min\{0.2, 0.4\} = 0.2 \end{aligned}$$

and 0 otherwise.

Definition 4.0.2 (Interpretation of predicates). *Let P be a set of predicate symbols. The interpretation of a predicate $p \in T_P(X)$ in a state $w \in W$ is given by the map*

$$\llbracket _ \rrbracket_w : T_P(X) \rightarrow \mathbf{A}$$

defined by

$$\llbracket p(t_1, \dots, t_n) \rrbracket_w = \sum_{i \in I} \left\{ \prod_{j=1}^n \llbracket t_j \rrbracket_w(r_j^i) \mid p(r_1^i, \dots, r_n^i) \right\}$$

where I is the cardinality of the set of all possible values $(r_1^i, \dots, r_n^i) \in \mathbb{R}^n$ satisfying $p(r_1^i, \dots, r_n^i)$, with each $p(r_1^i, \dots, r_n^i)$ being interpreted as a Boolean predicate.

Again this can be illustrated by computing the truth degree of predicate $x \leq 3$ in state w_2 , of Example 1.3.1, as the expression $\llbracket x \leq 3 \rrbracket_{w_2} = \llbracket x \rrbracket_{w_2}(3); \llbracket 3 \rrbracket_{w_2}(3)$. Now, instantiating with action lattices 2.2.7 and 2.2.11, this yields

G: $\min\{0.1, 1\} = 0.3$. The value 0.1 means that the predicate is true with a certainty 0.3.

R: $1.2 +_{\mathbf{R}} 3.7 = 4.9$. This interpretation corresponds to the energy consumed by evaluating the predicate.

Let us now interpret atomic programs. To do so, we first introduce the *programs weighting function*, given a set of programs Prog , and for each complete action lattice \mathbf{A} , $E^{\mathbf{A}} : \text{Prog} \rightarrow \mathbf{A}^{W \times W}$, which assigns a weight to a program execution, for any program $\pi \in \text{Prog}$.

Definition 4.0.3 (Interpretation of atomic programs). *The interpretation of atomic programs is a map*

$$\llbracket _ \rrbracket_0 : \text{Prog}_0 \rightarrow \mathbf{A}^{W \times W}$$

mapping each $x := t \in \text{Prog}_0$ into a function

$$\llbracket x := t \rrbracket_0(w, w') = \begin{cases} E^{\mathbf{A}}(x := t)(w, w') & \text{if } (w, w') \in (x := t)^M \\ \perp & \text{otherwise} \end{cases}$$

where $(x := t)$ is the standard relational semantics of a program assignment, typically given by:

$$(w, w') \in (x := t) \Leftrightarrow \begin{cases} w'(y)(r) = w(y)(r) & \text{if } y \neq x \\ w'(x)(r) = \llbracket t \rrbracket_w(r) & \text{otherwise} \end{cases}$$

It is important to highlight that the function E is sound in the sense that weights are only possible to assign to an admissible execution, i.e. iff $(w, w') \in (x := t)$. Intuitively, the expression $E^{\mathbf{A}}(\pi)(w_0, w_1)$

denotes the weighted execution of program π from state w_0 to w_1 , i.e. the weight associated to the corresponding transition. For instance, in Example 4.0.1, taking \mathbf{A} as a Godel algebra (Example 2.2.7), the expression $E^{\mathbf{G}}(x := 3)(w_0, w_1) = 0.6$ means that the system allows the execution of the assignment $x := 3$ from state w_0 to w_1 with truth degree 0.6. For simplification on the syntax, we will omit the parameter \mathbf{A} on the function $E^{\mathbf{A}}$. We illustrate the semantics by interpretation in the three distinct lattices 2.2.1, 2.2.7 and 2.2.11.

2: Consider states w_0 and w_1 defined as:

$$w_0(x)(r) = \begin{cases} \top, & \text{if } r = 1 \\ \perp & \text{otherwise} \end{cases} \quad w_1(x)(r) = \begin{cases} \top & \text{if } r = 2 \\ \perp, & \text{otherwise} \end{cases}$$

If $\llbracket x := 2 \rrbracket_0(w_0, w_1) = E(x := 2)(w_0, w_1) = \top$ and \perp otherwise, then $(w_0, w_1) \in \langle x := 2 \rangle$ and, conversely, if $(w_0, w_1) \in \langle x := 2 \rangle$ and $\llbracket x := 2 \rrbracket_0(w_0, w_1) = E(x := 2)(w_0, w_1) = \top$ and \perp otherwise, then the interpretation of the assignment $x := 2$ in $\Gamma(\mathbf{2})$ coincides with the classical scenario, where an action simply may or may not execute. In such case, the truth degree of execution is a Boolean value.

G: Assume $\llbracket x := 2 \rrbracket_0(w_0, w_1) = E(x := 2)(w_0, w_1) = 0.8$, $\llbracket x := x + y \rrbracket_0(w_1, w_2) = E(x := x + y)(w_1, w_2) = 0.4$, $\llbracket x := x + 1 \rrbracket_0(w'_2, w'_3) = E(x := x + 1)(w''_2, w''_3) = 0.7$ and $\llbracket y := y \times 2 \rrbracket_0(w'_2, w'_3) = E(y := y \times 2)(w''_2, w''_3) = 0.9$. These values are regarded as truth degrees, or, in a complementary reading, vagueness, associated to the execution of actions $x := 2$, $x := x + y$, $x := x + 1$ and $y := y \times 2$, respectively.

As a consequence of executing these assignments, the weights of the variables are updated accordingly. That is the case of x in state w_1 , by the value $w_1(x)(2) = \llbracket x \rrbracket_{w_0}(2) = 0.2$, and 0 otherwise, according to Definition 4.0.3. The weights of y are maintained, since the assignment $x := 2$ does not modify its value. The situation may be interpreted as follows: from a state where the evaluation of predicate $x = 1$ yields a truth degree 0.5 and $x = 2$ a truth degree 0.2, the execution of assignment $x := 2$ with a truth degree of 0.8, whenever occurs, leads to a state where $x = 2$ yields a truth degree 0.2. The weights of the variable x in w_2 are updated as follows:

$$\begin{aligned} w_2(x)(3) &= \llbracket x + y \rrbracket_{w_1}(3) = \llbracket x \rrbracket_{w_1}(2); \llbracket y \rrbracket_{w_1}(1) = \min\{1, 0.1\} = 0.1 \\ w_2(x)(4) &= \llbracket x + y \rrbracket_{w_1}(4) = \llbracket x \rrbracket_{w_1}(2); \llbracket y \rrbracket_{w_1}(2) = \min\{1, 0.4\} = 0.4 \end{aligned}$$

R: Consider, for example, $E(x := 2)(w_0, w_1) = 8$, $E(x := x + y)(w_1, w_2) = 4$, $E(x := x + 1)(w'_2, w'_3) = 7$ and $E(y := y \times 2)(w''_2, w''_3) = 9$. These values can be regarded as resources (e.g. energy) consumed by executing the associated assignments. Analogously to the previous case, the weights associated to y are maintained.

Finally, to interpret an arbitrary program in Prog the procedure is divided in two steps. First, the semantics of compound program constructs is given directly in terms of operations on \mathbf{A} -valued binary relations

$\mathbf{A}^{W \times W}$: union, composition, and Kleene closure. To interpret these operators, we define the following algebra:

Definition 4.0.4. Let $\mathbf{A} = (A, +, ;, *, \rightarrow, \cdot, 0, 1)$ be an complete action lattice and W be a finite set of states. The algebra of program weighting functions is the structure

$$\mathbf{E} = (Z(E), \cup, \circ, \emptyset, \chi, *)$$

where:

- $Z(E)$ is the universe of all the program weighting functions
- $(E(\pi_1) \cup E(\pi_2))(w, w') = E(\pi_1)(w, w') + E(\pi_2)(w, w')$
- $(E(\pi_1) \circ E(\pi_2))(w, w') = \sum_{w'' \in W} E(\pi_1)(w, w''); E(\pi_2)(w'', w')$
- $\emptyset(w, w') = \perp$
- $\chi(w, w') = \begin{cases} \top, & \text{if } w = w' \\ \perp, & \text{otherwise} \end{cases}$
-

$$\begin{aligned} (E(\pi))^*(w, w') &= \sum_{i \geq 0} (E(\pi))^i(w, w') \\ &= (E(\pi))^0(w, w') + (E(\pi))^1(w, w') + (E(\pi))^2(w, w') + \dots \end{aligned}$$

with $E(\pi_1), E(\pi_2) \in Z(E)$.

Note that operator $*$ can be defined as an infinite sum due to the completeness of the complete action lattice.

Theorem 4.0.1. The algebra of Definition 4.0.4 is a Kleene algebra.

Proof. Analogous to Theorem 3.4.2. □

Definition 4.0.5. The interpretation of a program $\pi \in \text{Prog}$ is a map

$$\llbracket _ \rrbracket : \text{Prog} \rightarrow \mathbf{A}^{W \times W}$$

recursively defined as

- $\llbracket \text{skip} \rrbracket = \chi$
- $\llbracket \pi_0 \rrbracket = \llbracket \pi_0 \rrbracket_0$, for each $\pi_0 \in \text{Prog}_0$

- $\llbracket \pi_1; \pi_2 \rrbracket = \llbracket \pi_1 \rrbracket \circ \llbracket \pi_2 \rrbracket$
- $\llbracket \pi_1 + \pi_2 \rrbracket = \llbracket \pi_1 \rrbracket \cup \llbracket \pi_2 \rrbracket$
- $\llbracket \pi^* \rrbracket = (\llbracket \pi \rrbracket)^*$.

where, for $r \in \mathbf{A}^{W \times W}$, $r^*(w, w') = \sum_{k \geq 0} r^k(w, w')$.

Again Example 1.3.1 illustrates choice and sequential composition by interpreting fragments $(x := 2); (x := x + y)$ and $(x := x + 1) + (y := y \times 2)$. The first one yields

$$\begin{aligned} \llbracket x := 2; x := x + y \rrbracket(w_0, w_2) &= (\llbracket x := 2 \rrbracket_0 \circ \llbracket x := x + y \rrbracket_0)(w_0, w_2) \\ &= \llbracket x := 2 \rrbracket_0(w_0, w_1); \llbracket x := x + y \rrbracket_0(w_1, w_2) \\ &= E(x := 2)(w_0, w_1); E(x := x + y)(w_1, w_2) \end{aligned}$$

which can be instantiated within the three usual lattices we have been considering:

2: Under this interpretation programs either fail or succeed. In the absence of failure execution proceeds sequentially; otherwise, if one (or both) statement fails (takes 'weight' \perp), so does the program.

G: In this case a truth degree is associated to the composition based on the corresponding degree for the atomic components. This is computed as a minimum. For example, if $E(x := 2)(w_0, w_1) = 0.8$ and $E(x := x + y)(w_1, w_2) = 0.4$ the overall truth degree for the composition becomes $\min\{0.8, 0.4\} = 0.4$.

R: Computations have a cost, under this interpretation, for example the amount of energy dissipated. Thus, $E(x := 2)(w_0, w_1); E(x := x + y)(w_1, w_2) = 8 +_{\mathbf{R}} 4 = 12$ represents the sum of the energy consumed by executing atomic programs $x := 2$ and $x := x + y$ sequentially.

The interpretation of the nondeterministic choice

$$(x := x + 1) + (y := y \times 2) \tag{59}$$

on the other hand, is given by

$$\begin{aligned} \llbracket (x := x + 1) + (y := y \times 2) \rrbracket(w'_2, w'_3) &= (\llbracket x := x + 1 \rrbracket_0 \cup \llbracket y := y \times 2 \rrbracket_0)(w'_2, w'_3) \\ &= \llbracket x := x + 1 \rrbracket_0(w'_2, w'_3) + \llbracket y := y \times 2 \rrbracket_0(w'_2, w'_3) \\ &= E(x := x + 1)(w'_2, w'_3) + E(y := y \times 2)(w'_2, w'_3) \\ &= E(x := x + 1)(w'_2, w'_3) + \perp \end{aligned}$$

and

$$\begin{aligned}
\llbracket (x := x + 1) + (y := y \times 2) \rrbracket (w''_2, w''_3) &= (\llbracket x := x + 1 \rrbracket_0 \cup \llbracket y := y \times 2 \rrbracket_0)(w''_2, w''_3) \\
&= \llbracket x := x + 1 \rrbracket_0(w''_2, w''_3) + \llbracket y := y \times 2 \rrbracket_0(w''_2, w''_3) \\
&= E(x := x + 1)(w''_2, w''_3) + E(y := y \times 2)(w''_2, w''_3) \\
&= \perp + E(y := y \times 2)(w''_2, w''_3)
\end{aligned}$$

Interpreting with the usual lattices, this yields

2: In this case choice is exactly nondeterministic choice: either one of $x := x + 1$ or $y := y \times 2$ is executed.

$$\begin{aligned}
- E(x := x + 1)(w'_2, w'_3) + \perp &= \top \vee \perp = \top \\
- \perp + E(y := y \times 2)(w'_2, w'_3) &= \perp \vee \top = \top
\end{aligned}$$

G: This interpretation yields the weighted choice between the alternative executions.

$$\begin{aligned}
- E(x := x + 1)(w'_2, w'_3) + \perp &= \max\{0.7, 0\} = 0.7. \\
- \perp + E(y := y \times 2)(w''_2, w''_3) &= \max\{0, 0.9\} = 0.9.
\end{aligned}$$

R: In this complete action lattice, the program is interpreted as the energy consumed by the two choices

$$\begin{aligned}
- E(x := x + 1)(w'_2, w'_3) + \perp &= \min\{7, +\infty\} = 7 \\
- \perp + E(y := y \times 2)(w''_2, w''_3) &= \min\{+\infty, 9\} = 9
\end{aligned}$$

Intuitively, $\llbracket (x := x + 1) + (y := y \times 2) \rrbracket$ represents the set of weighted alternative executions, formally a function which attributes weight $E(x := x + 1)(w'_2, w'_3)$ to the execution $x := x + 1$ and $E(y := y \times 2)(w''_2, w''_3)$ to the execution $y := y \times 2$.

In particular, if $E(x := x + 1)(w'_2, w'_3) + E(y := y \times 2)(w''_2, w''_3) = 1$, we may relate the **G** instantiations with the probabilistic choice $(x := x + 1) +_\alpha (y := y \times 2)$, where, as described in Section 1.4, assignment $x := x + 1$ is executed with probability α and $y := y \times 2$ with probability $1 - \alpha$. Thus we may attribute the weights of each assignment to the probabilities themselves, i.e. $E(x := x + 1)(w'_2, w'_3) = \alpha$ and $E(y := y \times 2)(w''_2, w''_3) = 1 - \alpha$. Additionally, if $E(x := x + 1)(w'_2, w'_3) + E(y := y \times 2)(w''_2, w''_3) \leq 1$, we may associate the semantics of (59), interpreted with **G**, with the sub-probabilistic choice $(\alpha)(x := x + 1) + (\beta)(y := y \times 2)$, by putting $E(x := x + 1)(w'_2, w'_3) = \alpha$ and $E(y := y \times 2)(w''_2, w''_3) = \beta$.

Note that nothing prevents the state space W from being infinite, because of completeness enforced upon \mathbf{A} . However, one may only compute explicitly a truth value associated with a program execution when W is finite.

DYNAMIC LOGICS FOR WEIGHTED *SINGLE-FLOW* COMPUTATIONS

Finally, we have now collected all the ingredients to focus on the logic. Each complete action lattice \mathbf{A} induces a multi-valued, equational dynamic logic $\Gamma(\mathbf{A})$ to reason about imperative programs interpreted as weighted “single-flow” computations with weights taken from \mathbf{A} . The proposed construction adapts the method introduced in [MNM16] by taking a non-empty set of variables (X) and terms constructed over X and \mathbb{R} in the syntax of the logic. The signature, formulæ, semantics and satisfaction relation of $\Gamma(\mathbf{A})$ are given below. In particular, the syntax and the semantics of programs are the ones presented in the previous chapter.

5.1 Generation of multi-valued equational dynamic logics

Once a language for programs is fixed (58), the set of formulas for $\Gamma(\mathbf{A})$ introduces, as expected, the universal and existential modalities over programs. Formally,

Definition 5.1.1. *A signature for $\Gamma(\mathbf{A})$ is a tuple*

$$\Delta = ((F, P), \Pi)$$

where (F, P) is a data signature composed by functional and predicate symbols, and $\Pi \subseteq \text{Prog}_0$. The set of formulas for Δ , denoted by $\text{Fm}^{\Gamma(\mathbf{A})}(\Delta)$, are the ones generated by the rule

$$\rho ::= \top \mid \perp \mid p(t_1, \dots, t_n) \mid \rho \vee \rho \mid \rho \wedge \rho \mid \rho \rightarrow \rho \mid \langle \pi \rangle \rho \mid [\pi] \rho$$

for $p(t_1, \dots, t_n) \in T_P(X)$ and π is a program in Prog .

Note that we sometimes make use of $\neg\rho$ as an abbreviation for $\rho \rightarrow \perp$, as in Example 1.3.1.

We can now turn to semantics. For each \mathbf{A} , models are defined over state spaces.

Definition 5.1.2 (Models). *Let $\Delta = ((F, P), \Pi)$ be a signature and X a set of variables. A $\Gamma(\mathbf{A})$ -model for Δ is a structure*

$$M = (W, V, E)$$

where

- $W \subseteq \mathbf{A}^{X \times \mathbb{R}}$ is a set of states;
- $V : P \times W \rightarrow \mathbf{A}$, defined as $V(p(t_1, \dots, t_n), w) = \llbracket p(t_1 \dots, t_n) \rrbracket_w$, for any $p(t_1, \dots, t_n) \in T_P(X)$;
- $E : \Pi \rightarrow \mathbf{A}^{W \times W}$ is a programs weighting function.

The set of $\Gamma(\mathbf{A})$ -models for Δ is denoted by $\text{Mod}^{\Gamma(\mathbf{A})}(\Delta)$.

An important element to care about when computing the semantics is the interpretation of tests. Our goal is to introduce a notion of a test in an arbitrary dynamic logic generated by \mathbf{A} . As mentioned above, tests are written as $\rho?$, for $\rho \in \text{Fm}^{\Gamma(\mathbf{A})}(\Delta)$. Their semantics resort, therefore, to the satisfaction relation for $\text{Fm}^{\Gamma(\mathbf{A})}(\Delta)$, which is defined as follows:

Definition 5.1.3 (Satisfaction). *Given a complete action lattice \mathbf{A} , the satisfaction relation for a model $M \in \text{Mod}^{\Gamma(\mathbf{A})}(\Delta)$ is given by*

$$\vDash_{\Gamma(\mathbf{A})} : W \times \text{Fm}^{\Gamma(\mathbf{A})}(\Delta) \rightarrow \mathbf{A}$$

recursively defined as follows:

- $(w \vDash_{\Gamma(\mathbf{A})} \top) = \top$
- $(w \vDash_{\Gamma(\mathbf{A})} \perp) = \perp$
- $V(p, w)$, for any $p(t_1, \dots, t_n) \in T_P(X)$
- $(w \vDash_{\Gamma(\mathbf{A})} \rho \wedge \rho') = (w \vDash_{\Gamma(\mathbf{A})} \rho) \cdot (w \vDash_{\Gamma(\mathbf{A})} \rho')$
- $(w \vDash_{\Gamma(\mathbf{A})} \rho \vee \rho') = (w \vDash_{\Gamma(\mathbf{A})} \rho) + (w \vDash_{\Gamma(\mathbf{A})} \rho')$
- $(w \vDash_{\Gamma(\mathbf{A})} \rho \rightarrow \rho') = (w \vDash_{\Gamma(\mathbf{A})} \rho) \rightarrow (w \vDash_{\Gamma(\mathbf{A})} \rho')$
- $(w \vDash_{\Gamma(\mathbf{A})} \langle \pi \rangle \rho) = \sum_{w' \in W} (\llbracket \pi \rrbracket(w, w'); (w' \vDash_{\Gamma(\mathbf{A})} \rho))$
- $(w \vDash_{\Gamma(\mathbf{A})} [\pi] \rho) = \bigwedge_{w' \in W} (\llbracket \pi \rrbracket(w, w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho))$

An axiomatic system for $\Gamma(\mathbf{A})$

A core ingredient of any logic is its axiomatisation. We discuss now an axiomatisation for $\Gamma(\mathbf{A})$, over a complete \mathbb{I} -action lattice. Similarly to [MNM16], such a characterisation is necessary to establish most of the results.

Let us establish first some auxiliary results.

Lemma 5.1.1. *The following properties hold in any complete \mathbb{I} -action lattice:*

$$a \leq b \Rightarrow a; c \leq b; c \quad (60)$$

$$a \leq b \ \& \ c \leq d \Rightarrow a + c \leq b + d \quad (61)$$

$$a \leq b \ \& \ c \leq d \Rightarrow a \cdot c \leq b \cdot d \quad (62)$$

$$a; (b \cdot c) \leq (a; b) \cdot (a; c) \quad (63)$$

$$a \leq b \Rightarrow (c \rightarrow a) \leq (c \rightarrow b) \quad (64)$$

$$a \leq b \Rightarrow (b \rightarrow c) \leq (a \rightarrow c) \quad (65)$$

$$a \rightarrow (b \rightarrow c) = (b; a) \rightarrow c \quad (66)$$

$$a \leq b \ \& \ a \leq c \Rightarrow a \leq b \cdot c \quad (67)$$

$$a \leq b \ \& \ c \leq d \Rightarrow a; c \leq b; d \quad (68)$$

If $a; b = b; a$,

$$a; a = a \Rightarrow (a \rightarrow (b \rightarrow c)); (a \rightarrow b) \leq (a \rightarrow c) \quad (69)$$

When I finite, we have also,

$$a \rightarrow \left(\bigwedge_{i \in I} b_i \right) = \bigwedge_{i \in I} (a \rightarrow b_i) \quad (70)$$

$$\left(\sum_{i \in I} a_i \right) \rightarrow b = \bigwedge_{i \in I} (a_i \rightarrow b) \quad (71)$$

$$\sum_{i \in I} (a_i \cdot b_i) \leq \sum_{i \in I} a_i \cdot \sum_{i \in I} b_i \quad (72)$$

Proof. The proof is analogous to [MNM16]. □

Lemma 5.1.2. *The following properties hold in any complete \mathbb{I} -action lattice:*

$$1. (1 \rightarrow a) = a$$

$$2. (\perp \rightarrow a) = 1$$

$$3. a \leq b \Leftrightarrow (a \rightarrow b) = 1$$

$$4. a = b \Leftrightarrow (a \leftrightarrow b) = 1$$

Proof. The proof of this lemma is analogous to [MNM16]. □

Lemma 5.1.3. *Let \mathbf{A} be a complete \mathbb{I} -action lattice. Then*

$$\bullet (w \vDash_{\Gamma(\mathbf{A})} \rho \rightarrow \rho') = \top \text{ iff } (w \vDash_{\Gamma(\mathbf{A})} \rho) \leq (w \vDash_{\Gamma(\mathbf{A})} \rho')$$

$$\bullet (w \vDash_{\Gamma(\mathbf{A})} \rho \leftrightarrow \rho') = \top \text{ iff } (w \vDash_{\Gamma(\mathbf{A})} \rho) = (w \vDash_{\Gamma(\mathbf{A})} \rho')$$

Proof. Since \mathbf{A} is a complete \mathbb{I} -action lattice, we have $\top = 1$ and hence we conclude both equivalences by clauses 3 and 4 of Lemma 5.1.2, respectively. \square

To prove axioms for $*$ -programs, we need also the following auxiliary result.

Theorem 5.1.1 ([Con12]). *Let $(A, +, ;, 0, 1, *)$ be a Kleene algebra. The following properties hold:*

$$a \leq a^* \tag{73}$$

$$a^* = a^{**} \tag{74}$$

$$a^* = a^*; a^* \tag{75}$$

$$1 + a; a^* = a^* \tag{76}$$

Now we prove the main axiomatisation for $\Gamma(\mathbf{A})$.

Lemma 5.1.4. *Let \mathbf{A} be a complete \mathbb{I} -action lattice. The following are valid formulæ in any $\Gamma(\mathbf{A})$:*

$$(1.) \quad \langle \pi \rangle (\rho \vee \rho') \leftrightarrow \langle \pi \rangle \rho \vee \langle \pi \rangle \rho'$$

$$(2.) \quad \langle \pi \rangle (\rho \wedge \rho') \rightarrow \langle \pi \rangle \rho \wedge \langle \pi \rangle \rho'$$

$$(3.) \quad \langle \pi + \pi' \rangle \rho \leftrightarrow \langle \pi \rangle \rho \vee \langle \pi' \rangle \rho$$

$$(4.) \quad \langle \pi; \pi' \rangle \rho \leftrightarrow \langle \pi \rangle \langle \pi' \rangle \rho$$

$$(5.) \quad \langle \pi \rangle \perp \leftrightarrow \perp$$

$$(6.) \quad \langle \pi \rangle \rho \rightarrow \langle \pi^* \rangle \rho$$

$$(7.) \quad \langle \pi^* \rangle \rho \leftrightarrow \langle \pi^*; \pi^* \rangle \rho$$

$$(8.) \quad \langle \pi^* \rangle \rho \leftrightarrow \langle \pi^{**} \rangle \rho$$

$$(9.) \quad \langle \pi^* \rangle \rho \leftrightarrow \rho \vee \langle \pi \rangle \langle \pi^* \rangle \rho$$

$$(10.) \quad [\pi + \pi'] \rho \leftrightarrow [\pi] \rho \wedge [\pi'] \rho$$

$$(11.) \quad [\pi] (\rho \wedge \rho') \leftrightarrow [\pi] \rho \wedge [\pi] \rho'$$

Proof. **(1.):**

$$\begin{aligned} & (w \vDash_{\Gamma(\mathbf{A})} \langle \pi \rangle (\rho \vee \rho')) \\ = & \quad \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\ & \sum_{w' \in W} ([\pi](w, w'); (w' \vDash_{\Gamma(\mathbf{A})} \rho \vee \rho')) \end{aligned}$$

$$\begin{aligned}
&= \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
&\quad \sum_{w' \in W} ((\llbracket \pi \rrbracket(w, w'); ((w' \vDash_{\Gamma(\mathbf{A})} \rho) + (w' \vDash_{\Gamma(\mathbf{A})} \rho')))) \\
&= \{ (10) \} \\
&\quad \sum_{w' \in W} ((\llbracket \pi \rrbracket(w, w'); (w' \vDash_{\Gamma(\mathbf{A})} \rho) + (\llbracket \pi \rrbracket(w, w'); (w' \vDash_{\Gamma(\mathbf{A})} \rho')))) \\
&= \{ \text{by (4) and (5)} \} \\
&\quad \sum_{w' \in W} (\llbracket \pi \rrbracket(w, w'); (w' \vDash_{\Gamma(\mathbf{A})} \rho)) + \sum_{w' \in W} (\llbracket \pi \rrbracket(w, w'); (w' \vDash_{\Gamma(\mathbf{A})} \rho')) \\
&= \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
&\quad (w \vDash_{\Gamma(\mathbf{A})} \langle \pi \rangle \rho) + (w \vDash_{\Gamma(\mathbf{A})} \langle \pi \rangle \rho') \\
&= \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
&\quad (w \vDash_{\Gamma(\mathbf{A})} \langle \pi \rangle \rho \vee \langle \pi \rangle \rho')
\end{aligned}$$

Therefore, by Lemma 5.1.3, $\langle \pi \rangle (\rho \vee \rho') \leftrightarrow \langle \pi \rangle \rho \vee \langle \pi \rangle \rho'$ is valid.

(2.):

$$\begin{aligned}
&(w \vDash_{\Gamma(\mathbf{A})} \langle \pi \rangle (\rho \wedge \rho')) \\
&= \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
&\quad \sum_{w' \in W} (\llbracket \pi \rrbracket(w, w'); (w' \vDash_{\Gamma(\mathbf{A})} \rho \wedge \rho')) \\
&= \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
&\quad \sum_{w' \in W} (\llbracket \pi \rrbracket(w, w'); ((w' \vDash_{\Gamma(\mathbf{A})} \rho) \cdot (w' \vDash_{\Gamma(\mathbf{A})} \rho')))) \\
&\leq \{ \text{by (63) and (61)} \} \\
&\quad \sum_{w' \in W} (\llbracket \pi \rrbracket(w, w'); (w' \vDash_{\Gamma(\mathbf{A})} \rho) \cdot (\llbracket \pi \rrbracket(w, w'); (w' \vDash_{\Gamma(\mathbf{A})} \rho')))) \\
&\leq \{ \text{by (72)} \} \\
&\quad \sum_{w' \in W} (\llbracket \pi \rrbracket(w, w'); (w' \vDash_{\Gamma(\mathbf{A})} \rho)) \cdot \sum_{w' \in W} (\llbracket \pi \rrbracket(w, w'); (w' \vDash_{\Gamma(\mathbf{A})} \rho')) \\
&= \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
&\quad (w \vDash_{\Gamma(\mathbf{A})} \langle \pi \rangle \rho) \cdot (w \vDash_{\Gamma(\mathbf{A})} \langle \pi \rangle \rho') \\
&= \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
&\quad (w \vDash_{\Gamma(\mathbf{A})} \langle \pi \rangle \rho \wedge \langle \pi \rangle \rho')
\end{aligned}$$

Therefore, by Lemma 5.1.3, $\langle \pi \rangle (\rho \wedge \rho') \rightarrow \langle \pi \rangle \rho \wedge \langle \pi \rangle \rho'$ is valid.

(3.):

$$\begin{aligned}
& (w \vDash_{\Gamma(\mathbf{A})} \langle \pi + \pi' \rangle \rho) \\
= & \quad \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
& \sum_{w' \in W} (\llbracket \pi + \pi' \rrbracket(w, w'); (w' \vDash_{\Gamma(\mathbf{A})} \rho)) \\
= & \quad \{ \text{definition of } \llbracket _ \rrbracket \} \\
& \sum_{w' \in W} ((\llbracket \pi \rrbracket(w, w') + \llbracket \pi' \rrbracket(w, w')); (w' \vDash \rho)) \\
= & \quad \{ \text{by (11)} \} \\
& \sum_{w' \in W} (\llbracket \pi \rrbracket(w, w'); (w' \vDash_{\Gamma(\mathbf{A})} \rho)) + \llbracket \pi' \rrbracket(w, w'); (w' \vDash_{\Gamma(\mathbf{A})} \rho)) \\
= & \quad \{ \text{by (4) and (5)} \} \\
& \sum_{w' \in W} (\llbracket \pi \rrbracket(w, w'); (w' \vDash_{\Gamma(\mathbf{A})} \rho)) + \sum_{w' \in W} (\llbracket \pi' \rrbracket(w, w'); (w' \vDash_{\Gamma(\mathbf{A})} \rho)) \\
= & \quad \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
& (w \vDash_{\Gamma(\mathbf{A})} \langle \pi \rangle \rho) + (w \vDash_{\Gamma(\mathbf{A})} \langle \pi' \rangle \rho) \\
= & \quad \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
& (w \vDash_{\Gamma(\mathbf{A})} \langle \pi \rangle \rho \vee \langle \pi' \rangle \rho)
\end{aligned}$$

Therefore, by Lemma 5.1.3, $\langle \pi + \pi' \rangle \rho \leftrightarrow \langle \pi \rangle \rho \vee \langle \pi' \rangle \rho$ is valid.

(4.):

$$\begin{aligned}
& (w \vDash_{\Gamma(\mathbf{A})} \langle \pi \rangle \langle \pi' \rangle \rho) \\
= & \quad \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
& \sum_{w' \in W} (\llbracket \pi \rrbracket(w, w'); (w \vDash_{\Gamma(\mathbf{A})} \langle \pi' \rangle \rho)) \\
= & \quad \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
& \sum_{w' \in W} \left(\llbracket \pi \rrbracket(w, w'); \sum_{w'' \in W} (\llbracket \pi' \rrbracket(w', w''); (w'' \vDash_{\Gamma(\mathbf{A})} \rho)) \right) \\
= & \quad \{ \text{by (10)} \} \\
& \sum_{w' \in W} \left(\sum_{w'' \in W} (\llbracket \pi \rrbracket(w, w'); (\llbracket \pi' \rrbracket(w', w''); (w'' \vDash_{\Gamma(\mathbf{A})} \rho))) \right) \\
= & \quad \{ \text{by (4) and (5)} \} \\
& \sum_{w'' \in W} \left(\sum_{w' \in W} (\llbracket \pi \rrbracket(w, w'); \llbracket \pi' \rrbracket(w', w''); (w'' \vDash_{\Gamma(\mathbf{A})} \rho)) \right) \\
= & \quad \{ (w'' \vDash_{\Gamma(\mathbf{A})} \rho) \text{ is independent of } w' \} \\
& \sum_{w'' \in W} \left(\sum_{w' \in W} (\llbracket \pi \rrbracket(w, w'); \llbracket \pi' \rrbracket(w', w'')); (w'' \vDash_{\Gamma(\mathbf{A})} \rho) \right) \\
= & \quad \{ \text{definition of } \circ \} \\
& \sum_{w'' \in W} (\llbracket \pi; \pi' \rrbracket(w, w''); (w'' \vDash_{\Gamma(\mathbf{A})} \rho)) \\
= & \quad \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
& (w \vDash_{\Gamma(\mathbf{A})} \langle \pi; \pi' \rangle \rho)
\end{aligned}$$

Therefore, by Lemma 5.1.3, $\langle \pi \rangle \langle \pi' \rangle \rho \leftrightarrow \langle \pi; \pi' \rangle \rho$ is valid.

(5.):

$$\begin{aligned}
& (w \vDash_{\Gamma(\mathbf{A})} \langle \pi \rangle \perp) \\
= & \quad \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
& \sum_{w' \in W} (\llbracket \pi \rrbracket(w, w'); (w' \vDash_{\Gamma(\mathbf{A})} \perp)) \\
= & \quad \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
& \sum_{w' \in W} (\llbracket \pi \rrbracket(w, w'); \perp) \\
= & \quad \{ \text{by (12)} \} \\
& \sum_{w' \in W} (\perp) \\
= & \quad \{ \text{by (40)} \} \\
& \perp
\end{aligned}$$

Therefore, by Lemma 5.1.3, $\langle \pi \rangle \perp \leftrightarrow \perp$ is valid.

Since $\mathbf{E} = (Z(E), \cup, \circ, \emptyset, \chi, ^*)$ is a Kleene algebra, by Theorem 5.1.1 and (73)–(76), we have:

$$\llbracket \pi \rrbracket(w, w') \leq \llbracket \pi^* \rrbracket(w, w') \quad (77)$$

$$\llbracket \pi^* \rrbracket(w, w') = \llbracket \pi^{**} \rrbracket(w, w') \quad (78)$$

$$\llbracket \pi^* \rrbracket(w, w') = \llbracket \pi^*; \pi^* \rrbracket(w, w') \quad (79)$$

$$\llbracket \mathbf{skip} + \pi; \pi^* \rrbracket(w, w') = \llbracket \pi^* \rrbracket(w, w') \quad (80)$$

(6.):

for any $w' \in W$ $\llbracket \pi \rrbracket(w, w') \leq \llbracket \pi^* \rrbracket(w, w')$

\Rightarrow { by (60) }

for any $w' \in W$ $\llbracket \pi \rrbracket(w, w'); (w' \vDash_{\Gamma(\mathbf{A})} \rho) \leq \llbracket \pi^* \rrbracket(w, w'); (w' \vDash_{\Gamma(\mathbf{A})} \rho)$

\Rightarrow { by (61) }

$$\sum_{w' \in W} (\llbracket \pi \rrbracket(w, w'); (w' \vDash_{\Gamma(\mathbf{A})} \rho)) \leq \sum_{w' \in W} (\llbracket \pi^* \rrbracket(w, w'); (w' \vDash_{\Gamma(\mathbf{A})} \rho))$$

\Leftrightarrow { definition of $\vDash_{\Gamma(\mathbf{A})}$ }

$$(w \vDash_{\Gamma(\mathbf{A})} \langle \pi \rangle \rho) \leq (w \vDash_{\Gamma(\mathbf{A})} \langle \pi^* \rangle \rho)$$

Since by (77), $\llbracket \pi \rrbracket(w, w') \leq \llbracket \pi^* \rrbracket(w, w')$ holds for any $w, w' \in W$, we conclude $(w \vDash_{\Gamma(\mathbf{A})} \langle \pi \rangle \rho) \leq (w \vDash_{\Gamma(\mathbf{A})} \langle \pi^* \rangle \rho)$. Hence, by Lemma 5.1.3, $\langle \pi \rangle \rho \rightarrow \langle \pi^* \rangle \rho$ is valid. The remaining of the first two proofs follows exactly the same steps but starting from (78) and (79).

(9.):

$$\begin{aligned}
& \llbracket \mathbf{skip} + \pi; \pi^* \rrbracket(w, w') = \llbracket \pi^* \rrbracket(w, w') \text{ for any } w, w' \in W \\
\Leftrightarrow & \quad \{ \text{definition of } \llbracket _ \rrbracket \} \\
& \llbracket \mathbf{skip} \rrbracket(w, w') + \llbracket \pi; \pi^* \rrbracket(w, w') = \llbracket \pi^* \rrbracket(w, w') \text{ for any } w' \in W \\
\Leftrightarrow & \quad \{ a = b \Rightarrow a; c = b; c \} \\
& (\llbracket \mathbf{skip} \rrbracket(w, w') + \llbracket \pi; \pi^* \rrbracket(w, w')) ; (w' \vDash_{\Gamma(\mathbf{A})} \rho) = \llbracket \pi^* \rrbracket(w, w') ; (w' \vDash_{\Gamma(\mathbf{A})} \rho) \\
& \text{for any } w' \in W \\
\Leftrightarrow & \quad \{ (11) \} \\
& \llbracket \mathbf{skip} \rrbracket(w, w') ; (w' \vDash_{\Gamma(\mathbf{A})} \rho) + \llbracket \pi; \pi^* \rrbracket(w, w') ; (w' \vDash_{\Gamma(\mathbf{A})} \rho) \\
& = \llbracket \pi^* \rrbracket(w, w') ; (w' \vDash_{\Gamma(\mathbf{A})} \rho), \text{ for any } w' \in W \\
\Leftrightarrow & \quad \{ a_i = b_i, i \in I \Rightarrow \sum_{i \in I} a_i = \sum_{i \in I} b_i \} \\
& \sum_{w' \in W} (\llbracket \mathbf{skip} \rrbracket(w, w') ; (w' \vDash_{\Gamma(\mathbf{A})} \rho) + \llbracket \pi; \pi^* \rrbracket(w, w') ; (w' \vDash_{\Gamma(\mathbf{A})} \rho)) \\
& = \sum_{w' \in W} (\llbracket \pi^* \rrbracket(w, w') ; (w' \vDash_{\Gamma(\mathbf{A})} \rho)) \\
\Leftrightarrow & \quad \{ \text{by (4) and (5)} \} \\
& \sum_{w' \in W} (\llbracket \mathbf{skip} \rrbracket(w, w') ; (w' \vDash_{\Gamma(\mathbf{A})} \rho)) + \sum_{w' \in W} (\llbracket \pi; \pi^* \rrbracket(w, w') ; (w' \vDash_{\Gamma(\mathbf{A})} \rho)) \\
& = \sum_{w' \in W} (\llbracket \pi^* \rrbracket(w, w') ; (w' \vDash_{\Gamma(\mathbf{A})} \rho)) \\
\Leftrightarrow & \quad \{ \text{definition of } \llbracket _ \rrbracket, (9) \text{ and definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
& (w \vDash_{\Gamma(\mathbf{A})} \rho) + (w \vDash_{\Gamma(\mathbf{A})} \langle \pi; \pi^* \rangle \rho) = (w \vDash_{\Gamma(\mathbf{A})} \langle \pi^* \rangle \rho) \\
\Leftrightarrow & \quad \{ (4.) \} \\
& (w \vDash_{\Gamma(\mathbf{A})} \rho) + (w \vDash_{\Gamma(\mathbf{A})} \langle \pi \rangle \langle \pi^* \rangle \rho) = (w \vDash_{\Gamma(\mathbf{A})} \langle \pi^* \rangle \rho) \\
\Leftrightarrow & \quad \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
& (w \vDash_{\Gamma(\mathbf{A})} \rho \vee \langle \pi \rangle \langle \pi^* \rangle \rho) = (w \vDash_{\Gamma(\mathbf{A})} \langle \pi^* \rangle \rho)
\end{aligned}$$

Therefore, by Lemma 5.1.3, $\langle \pi^* \rangle \rho \leftrightarrow \rho \vee \langle \pi \rangle \langle \pi^* \rangle \rho$ holds.

(10.):

$$\begin{aligned}
& (w \vDash_{\Gamma(\mathbf{A})} [\pi + \pi']\rho) \\
= & \quad \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
& \bigwedge_{w' \in W} (\llbracket \pi + \pi' \rrbracket(w, w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho)) \\
= & \quad \{ \text{definition of } \llbracket _ \rrbracket \} \\
& \bigwedge_{w' \in W} ((\llbracket \pi \rrbracket(w, w') + \llbracket \pi' \rrbracket(w, w')) \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho)) \\
= & \quad \{ (71) \} \\
& \bigwedge_{w' \in W} ((\llbracket \pi \rrbracket(w, w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho)) \cdot (\llbracket \pi' \rrbracket(w, w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho))) \\
= & \quad \{ (19) \} \\
& \bigwedge_{w' \in W} (\llbracket \pi \rrbracket(w, w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho)) \cdot \bigwedge_{w' \in W} (\llbracket \pi' \rrbracket(w, w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho)) \\
= & \quad \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
& (w \vDash_{\Gamma(\mathbf{A})} [\pi]\rho) \cdot (w \vDash_{\Gamma(\mathbf{A})} [\pi']\rho) \\
= & \quad \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
& (w \vDash_{\Gamma(\mathbf{A})} [\pi]\rho \wedge [\pi']\rho)
\end{aligned}$$

Therefore, by Lemma 5.1.3, we have that $[\pi + \pi']\rho \leftrightarrow [\pi]\rho \wedge [\pi']\rho$ is valid.

(11.):

$$\begin{aligned}
& (w \vDash_{\Gamma(\mathbf{A})} [\pi](\rho \wedge \rho')) \\
= & \quad \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
& \bigwedge_{w' \in W} ([\pi](w, w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho \wedge \rho')) \\
= & \quad \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
& \bigwedge_{w' \in W} ([\pi](w, w') \rightarrow ((w' \vDash_{\Gamma(\mathbf{A})} \rho) \cdot (w' \vDash_{\Gamma(\mathbf{A})} \rho'))) \\
= & \quad \{ (70) \} \\
& \bigwedge_{w' \in W} \left(([\pi](w, w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho)) \cdot ([\pi](w, w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho')) \right) \\
= & \quad \{ (19) \} \\
& \bigwedge_{w' \in W} ([\pi](w, w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho)) \cdot \bigwedge_{w' \in W} ([\pi](w, w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho')) \\
= & \quad \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
& (w \vDash_{\Gamma(\mathbf{A})} [\pi]\rho) \cdot (w \vDash_{\Gamma(\mathbf{A})} [\pi]\rho') \\
= & \quad \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
& (w \vDash [\pi]\rho \wedge [\pi]\rho')
\end{aligned}$$

Therefore, by Lemma 5.1.3, $[\pi](\rho \wedge \rho') \leftrightarrow [\pi]\rho \wedge [\pi]\rho'$ holds.

□

Lemma 5.1.5. *Let \mathbf{A} be a complete \mathbb{I} -action lattice satisfying*

$$a \rightarrow (b \rightarrow c) = (a; b) \rightarrow c \quad (81)$$

Then property

$$[\pi; \pi']\rho \leftrightarrow [\pi][\pi']\rho \quad (82)$$

is valid in $\Gamma(\mathbf{A})$.

Proof.

$$\begin{aligned}
& (w \vDash_{\Gamma(\mathbf{A})} [\pi; \pi'] \rho) \\
= & \quad \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
& \bigwedge_{w' \in W} ([\pi; \pi'](w, w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho)) \\
= & \quad \{ \text{definition of } [_] \} \\
& \bigwedge_{w' \in W} \left(\left(\sum_{w'' \in W} ([\pi](w, w''); [\pi'](w'', w')) \right) \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho) \right) \\
= & \quad \{ \text{by (71)} \} \\
& \bigwedge_{w' \in W} \left(\bigwedge_{w'' \in W} ([\pi](w, w''); [\pi'](w'', w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho)) \right) \\
= & \quad \{ \text{by (81)} \} \\
& \bigwedge_{w' \in W} \left(\bigwedge_{w'' \in W} ([\pi](w, w'') \rightarrow ([\pi'](w'', w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho))) \right) \\
= & \quad \{ \text{by (19) and (20)} \} \\
& \bigwedge_{w'' \in W} \left(\bigwedge_{w' \in W} ([\pi](w, w'') \rightarrow ([\pi'](w'', w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho))) \right) \\
= & \quad \{ \text{by (70)} \} \\
& \bigwedge_{w'' \in W} \left([\pi](w, w'') \rightarrow \left(\bigwedge_{w' \in W} ([\pi'](w'', w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho)) \right) \right) \\
= & \quad \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
& \bigwedge_{w'' \in W} ([\pi](w, w'') \rightarrow (w'' \vDash_{\Gamma(\mathbf{A})} [\pi'] \rho)) \\
= & \quad \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
& (w \vDash_{\Gamma(\mathbf{A})} [\pi][\pi'] \rho)
\end{aligned}$$

□

Lemma 5.1.6. *Let \mathbf{A} be a complete \mathbb{I} -action lattice satisfying*

$$a; b = b; a \tag{83}$$

$$a; a = a \tag{84}$$

Then property

$$[\pi](\rho \rightarrow \rho') \rightarrow ([\pi]\rho \rightarrow [\pi]\rho') \tag{85}$$

is valid in $\Gamma(\mathbf{A})$.

Proof. In order to prove (85) observe that

$$\begin{aligned}
& (w \vDash_{\Gamma(\mathbf{A})} [\pi](\rho \rightarrow \rho')); (w \vDash_{\Gamma(\mathbf{A})} [\pi]\rho) \\
= & \bigwedge_{w' \in W} ([\pi](w, w') \rightarrow ((w' \vDash_{\Gamma(\mathbf{A})} \rho) \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho'))); \bigwedge_{w' \in W} ([\pi](w, w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho))
\end{aligned}$$

By (68) we have:

$$\bigwedge_{w' \in W} (\llbracket \pi \rrbracket(w, w') \rightarrow ((w' \vDash_{\Gamma(\mathbf{A})} \rho) \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho'))); \bigwedge_{w' \in W} (\llbracket \pi \rrbracket(w, w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho)) \\ \leq (\llbracket \pi \rrbracket(w, w') \rightarrow ((w' \vDash_{\Gamma(\mathbf{A})} \rho) \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho'))); (\llbracket \pi \rrbracket(w, w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho))$$

for any $w' \in W$. Moreover, we have for any $w' \in W$,

$$(\llbracket \pi \rrbracket(w, w') \rightarrow ((w' \vDash_{\Gamma(\mathbf{A})} \rho) \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho'))); (\llbracket \pi \rrbracket(w, w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho)) \\ \leq \quad \{ \text{by hypothesis and (69)} \} \\ \llbracket \pi \rrbracket(w, w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho')$$

Therefore, by (67) we have that

$$\bigwedge_{w' \in W} (\llbracket \pi \rrbracket(w, w') \rightarrow ((w' \vDash_{\Gamma(\mathbf{A})} \rho) \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho'))); \bigwedge_{w' \in W} (\llbracket \pi \rrbracket(w, w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho)) \\ \leq \bigwedge_{w' \in W} (\llbracket \pi \rrbracket(w, w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho')) = (w \vDash_{\Gamma(\mathbf{A})} [\pi]\rho')$$

Moreover,

$$(w \vDash_{\Gamma(\mathbf{A})} [\pi](\rho \rightarrow \rho')); (w \vDash_{\Gamma(\mathbf{A})} [\pi]\rho) \leq (w \vDash_{\Gamma(\mathbf{A})} [\pi]\rho') \\ \Leftrightarrow \quad \{ (83) \} \\ (w \vDash_{\Gamma(\mathbf{A})} [\pi]\rho); (w \vDash_{\Gamma(\mathbf{A})} [\pi](\rho \rightarrow \rho')) \leq (w \vDash_{\Gamma(\mathbf{A})} [\pi]\rho') \\ \Leftrightarrow \quad \{ \text{by (17)} \} \\ (w \vDash_{\Gamma(\mathbf{A})} [\pi](\rho \rightarrow \rho')) \leq (w \vDash_{\Gamma(\mathbf{A})} [\pi]\rho) \rightarrow (w \vDash_{\Gamma(\mathbf{A})} [\pi]\rho') \\ \Leftrightarrow \quad \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\ (w \vDash_{\Gamma(\mathbf{A})} [\pi](\rho \rightarrow \rho')) \leq (w \vDash_{\Gamma(\mathbf{A})} [\pi]\rho \rightarrow [\pi]\rho')$$

Therefore, by Lemma 5.1.3, $[\pi](\rho \rightarrow \rho') \rightarrow ([\pi]\rho \rightarrow [\pi]\rho')$.

□

Lemma 5.1.7. *Let \mathbf{A} be a complete \mathbb{H} -action lattice. The following are valid formulæ in $\Gamma(\mathbf{A})$:*

$$(12.) \quad \rho \wedge [\pi][\pi^*]\rho \leftrightarrow [\pi^*]\rho$$

$$(13.) \quad [\pi^*](\rho \rightarrow [\pi]\rho) \rightarrow (\rho \rightarrow [\pi^*]\rho)$$

Proof. **(12.):**

$$\llbracket \text{skip} + \pi; \pi^* \rrbracket(w, w') = \llbracket \pi^* \rrbracket(w, w') \text{ for any } w, w' \in W \\ \Leftrightarrow \quad \{ a = b \Rightarrow (a \rightarrow c) = (b \rightarrow c) \}$$

$$\begin{aligned}
& (\llbracket \mathbf{skip} + \pi; \pi^* \rrbracket(w, w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho)) \\
& = (\llbracket \pi^* \rrbracket(w, w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho)) \text{ for any } w' \in W \\
\Leftrightarrow & \quad \{ a_i = b_i, i \in I \Rightarrow \bigwedge_{i \in I} a_i = \bigwedge_{i \in I} b_i \} \\
& \bigwedge_{w' \in W} (\llbracket \mathbf{skip} + \pi; \pi^* \rrbracket(w, w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho)) = \bigwedge_{w' \in W} (\llbracket \pi^* \rrbracket(w, w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho)) \\
\Leftrightarrow & \quad \{ \text{definition of } \llbracket _ \rrbracket \} \\
& \bigwedge_{w' \in W} ((\llbracket \mathbf{skip} \rrbracket(w, w') + \llbracket \pi; \pi^* \rrbracket(w, w')) \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho)) \\
& = \bigwedge_{w' \in W} (\llbracket \pi^* \rrbracket(w, w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho)) \\
\Leftrightarrow & \quad \{ \text{by (71)} \} \\
& \bigwedge_{w' \in W} ((\llbracket \mathbf{skip} \rrbracket(w, w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho)) \cdot (\llbracket \pi; \pi^* \rrbracket(w, w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho))) \\
& = \bigwedge_{w' \in W} (\llbracket \pi^* \rrbracket(w, w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho)) \\
\Leftrightarrow & \quad \{ \text{by (19) and (20)} \} \\
& \bigwedge_{w' \in W} ((\llbracket \mathbf{skip} \rrbracket(w, w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho)) \cdot \bigwedge_{w' \in W} ((\llbracket \pi; \pi^* \rrbracket(w, w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho)))) \\
& = \bigwedge_{w' \in W} (\llbracket \pi^* \rrbracket(w, w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho)) \\
\Leftrightarrow & \quad \{ \text{step } \star \} \\
& (w \vDash_{\Gamma(\mathbf{A})} \rho) \cdot \bigwedge_{w' \in W} (\llbracket \pi; \pi^* \rrbracket(w, w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho)) \\
& = \bigwedge_{w' \in W} (\llbracket \pi^* \rrbracket(w, w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho)) \\
\Leftrightarrow & \quad \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
& (w \vDash_{\Gamma(\mathbf{A})} \rho) \cdot (w \vDash_{\Gamma(\mathbf{A})} [\pi; \pi^*]\rho) = (w \vDash_{\Gamma(\mathbf{A})} [\pi^*]\rho) \\
\Leftrightarrow & \quad \{ (82) \} \\
& (w \vDash_{\Gamma(\mathbf{A})} \rho) \cdot (w \vDash_{\Gamma(\mathbf{A})} [\pi][\pi^*]\rho) = (w \vDash_{\Gamma(\mathbf{A})} [\pi^*]\rho) \\
\Leftrightarrow & \quad \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
& (w \vDash_{\Gamma(\mathbf{A})} \rho \wedge [\pi][\pi^*]\rho) = (w \vDash_{\Gamma(\mathbf{A})} [\pi^*]\rho)
\end{aligned}$$

The proof step annotated with \star comes from

$$\begin{aligned}
& \bigwedge_{w' \in W} ((\llbracket \mathbf{skip} \rrbracket(w, w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho)) \\
\Leftrightarrow & \quad \{ \text{definition of } \llbracket \mathbf{skip} \rrbracket \}
\end{aligned}$$

$$\begin{aligned}
& (1 \rightarrow (w \vDash_{\Gamma(\mathbf{A})} \rho)) \cdot \bigwedge_{w' \in W \{w\}} (\perp \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho)) \\
\Leftrightarrow & \quad \{ \text{by Lemma 5.1.2} \} \\
& (w \vDash_{\Gamma(\mathbf{A})} \rho) \cdot \bigwedge_{w' \in W \{w\}} (\top) \\
\Leftrightarrow & \quad \{ \text{by (21)} \} \\
& (w \vDash_{\Gamma(\mathbf{A})} \rho) \cdot \top \\
\Leftrightarrow & \quad \{ a \leq \top \} \\
& (w \vDash_{\Gamma(\mathbf{A})} \rho)
\end{aligned}$$

Therefore, since by (80), $\llbracket \mathbf{skip} + \pi; \pi^* \rrbracket(w, w') = \llbracket \pi^* \rrbracket(w, w')$ for any $w, w' \in W$, we have $(w \vDash \rho \wedge [\pi][\pi^*]\rho) = (w \vDash [\pi^*]\rho)$. By Lemma 5.1.3, $\rho \wedge [\pi][\pi^*]\rho \leftrightarrow [\pi^*]\rho$ is valid.

(13.):

$$\begin{aligned}
& (w \vDash_{\Gamma(\mathbf{A})} [\pi^*](\rho \rightarrow [\pi]\rho)) \\
= & \quad \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
& \bigwedge_{w' \in W} ([\pi^*](w, w') \rightarrow ((w' \vDash_{\Gamma(\mathbf{A})} \rho) \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} [\pi]\rho))) \\
= & \quad \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
& \bigwedge_{w' \in W} ([\pi^*](w, w') \rightarrow ((w' \vDash_{\Gamma(\mathbf{A})} \rho) \rightarrow (\bigwedge_{w'' \in W} ([\pi](w', w'') \rightarrow (w'' \vDash_{\Gamma(\mathbf{A})} \rho)))))) \\
= & \quad \{ (70) \text{ twice} \} \\
& \bigwedge_{w' \in W} \bigwedge_{w'' \in W} ([\pi^*](w, w') \rightarrow ((w' \vDash_{\Gamma(\mathbf{A})} \rho) \rightarrow ([\pi](w', w'') \rightarrow (w'' \vDash_{\Gamma(\mathbf{A})} \rho)))) \\
= & \quad \{ (66) \text{ 3}\times \text{ and } (83) \text{ from } \mathbb{H} \} \\
& \bigwedge_{w' \in W} \bigwedge_{w'' \in W} ((w' \vDash_{\Gamma(\mathbf{A})} \rho) \rightarrow (([\pi^*](w, w'); [\pi](w', w'')) \rightarrow (w'' \vDash_{\Gamma(\mathbf{A})} \rho))) \\
\leq & \quad \{ \text{step } ** \text{ and } (62) \} \\
& \bigwedge_{w' \in W} \bigwedge_{w'' \in W} ((w' \vDash_{\Gamma(\mathbf{A})} \rho) \rightarrow ([\pi^*](w, w'') \rightarrow (w'' \vDash_{\Gamma(\mathbf{A})} \rho))) \\
= & \quad \{ \text{by } (70) \} \\
& \bigwedge_{w' \in W} ((w' \vDash_{\Gamma(\mathbf{A})} \rho) \rightarrow (\bigwedge_{w'' \in W} ([\pi^*](w, w'') \rightarrow (w'' \vDash_{\Gamma(\mathbf{A})} \rho)))) \\
= & \quad \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
& \bigwedge_{w' \in W} ((w' \vDash_{\Gamma(\mathbf{A})} \rho) \rightarrow (w \vDash_{\Gamma(\mathbf{A})} [\pi^*]\rho)) \\
\leq & \quad \{ \text{infimum} \} \\
& (w \vDash_{\Gamma(\mathbf{A})} \rho) \rightarrow (w \vDash_{\Gamma(\mathbf{A})} [\pi^*]\rho) \\
= & \quad \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
& (w \vDash_{\Gamma(\mathbf{A})} \rho \rightarrow [\pi^*]\rho)
\end{aligned}$$

Step ** holds by definition of program interpretation and (37):

$$\begin{aligned}
& [\pi^*](w, w'); [\pi](w', w'') \leq \sum_{w''' \in W} ([\pi^*](w, w'''); [\pi](w''', w'')) = [\pi^*; \pi](w, w'') \\
\Rightarrow & \quad \{ \text{by } (65) \} \\
& [\pi^*; \pi](w, w'') \rightarrow (w'' \vDash_{\Gamma(\mathbf{A})} \rho) \leq [\pi^*](w, w'); [\pi](w', w'') \rightarrow (w'' \vDash_{\Gamma(\mathbf{A})} \rho) \\
\Rightarrow & \quad \{ \text{by } (64) \} \\
& (w' \vDash_{\Gamma(\mathbf{A})} \rho) \rightarrow ([\pi^*; \pi](w, w'') \rightarrow (w'' \vDash_{\Gamma(\mathbf{A})} \rho)) \\
& \leq (w' \vDash_{\Gamma(\mathbf{A})} \rho) \rightarrow ([\pi^*](w, w'); [\pi](w', w'') \rightarrow (w'' \vDash_{\Gamma(\mathbf{A})} \rho))
\end{aligned}$$

□

Lemma 5.1.8. *Let \mathbf{A} be a complete \mathbb{I} -action lattice. Then*

$$\langle \pi \rangle \rho \rightarrow \perp \leftrightarrow [\pi](\rho \rightarrow \perp) \quad (86)$$

is valid in $\Gamma(\mathbf{A})$.

Proof.

$$\begin{aligned}
& (w \vDash_{\Gamma(\mathbf{A})} \langle \pi \rangle \rho \rightarrow \perp) \\
= & \quad \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
& (w \vDash_{\Gamma(\mathbf{A})} \langle \pi \rangle \rho) \rightarrow (w \vDash_{\Gamma(\mathbf{A})} \perp) \\
= & \quad \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
& \left(\sum_{w' \in W} (\llbracket \pi \rrbracket(w, w'); (w' \vDash_{\Gamma(\mathbf{A})} \rho)) \right) \rightarrow (w \vDash_{\Gamma(\mathbf{A})} \perp) \\
= & \quad \{ \text{by (71) and} \\
& \quad (w' \vDash_{\Gamma(\mathbf{A})} \perp) = \perp = (w \vDash_{\Gamma(\mathbf{A})} \perp) \} \\
& \bigwedge_{w' \in W} (\llbracket \pi \rrbracket(w, w') \rightarrow ((w' \vDash_{\Gamma(\mathbf{A})} \rho) \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \perp))) \\
= & \quad \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
& \bigwedge_{w' \in W} (\llbracket \pi \rrbracket(w, w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho \rightarrow \perp)) \\
= & \quad \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
& (w \vDash_{\Gamma(\mathbf{A})} [\pi](\rho \rightarrow \perp))
\end{aligned}$$

Therefore, we conclude the validity of (86) by Lemma 5.1.3. □

5.2 Back to the weighted *single-flow* semantics

The interpretation of tests in the classical, Boolean case is given by co-reflexive relations $R_{\rho?} = \{(w, w) \mid w \vDash_{\Gamma(\mathbf{A})} \rho\}$. In the generic setting of the present work this generalises to

$$\llbracket \rho? \rrbracket(w, w') = \begin{cases} (w \vDash_{\Gamma(\mathbf{A})} \rho) & \text{if } w = w' \\ \perp & \text{otherwise} \end{cases}$$

Let us revisit Example 1.3.1 to interpret the conditional statement

$$\mathbf{if } x \leq 3 \mathbf{ then } x := x + 1 \mathbf{ else } y := y \times 2 \quad (87)$$

translated to $((x \leq 3)?; x := x + 1) + (((x \leq 3) \rightarrow \perp)?; y := y \times 2)$. Using the value computed for predicate $x \leq 3$, this leads to

$$\begin{aligned}
& \llbracket ((x \leq 3)?; x := x + 1) + (((x \leq 3) \rightarrow \perp)?; y := y \times 2) \rrbracket (w'_2, w'_3) = \\
& = (\llbracket (x \leq 3)? \rrbracket \circ \llbracket x := x + 1 \rrbracket \cup \llbracket ((x \leq 3) \rightarrow \perp)? \rrbracket \circ \llbracket y := y \times 2 \rrbracket) (w'_2, w'_3) \\
& = \llbracket (x \leq 3)? \rrbracket (w'_2, w'_2); \llbracket x := x + 1 \rrbracket (w'_2, w'_3) + \\
& \quad \llbracket ((x \leq 3) \rightarrow \perp)? \rrbracket (w'_2, w'_2); \llbracket y := y \times 2 \rrbracket (w'_2, w'_3) \\
& = (w'_2 \vDash_{\Gamma(\mathbf{A})} x \leq 3); E(x := x + 1)(w'_2, w'_3) + \\
& \quad (w'_2 \vDash_{\Gamma(\mathbf{A})} (x \leq 3) \rightarrow \perp); E(y := y \times 2)(w'_2, w'_3) \\
& = (w'_2 \vDash_{\Gamma(\mathbf{A})} x \leq 3); E(x := x + 1)(w'_2, w'_3) + \perp
\end{aligned}$$

$$\begin{aligned}
& \llbracket ((x \leq 3)?; x := x + 1) + (((x \leq 3) \rightarrow \perp)?; y := y \times 2) \rrbracket (w''_2, w''_3) = \\
& = (\llbracket (x \leq 3)? \rrbracket \circ \llbracket x := x + 1 \rrbracket \cup \llbracket ((x \leq 3) \rightarrow \perp)? \rrbracket \circ \llbracket y := y \times 2 \rrbracket) (w''_2, w''_3) \\
& = \llbracket (x \leq 3)? \rrbracket (w''_2, w''_2); \llbracket x := x + 1 \rrbracket (w''_2, w''_3) + \\
& \quad \llbracket ((x \leq 3) \rightarrow \perp)? \rrbracket (w''_2, w''_2); \llbracket y := y \times 2 \rrbracket (w''_2, w''_3) \\
& = (w''_2 \vDash_{\Gamma(\mathbf{A})} x \leq 3); E(x := x + 1)(w''_2, w''_3) + \\
& \quad (w''_2 \vDash_{\Gamma(\mathbf{A})} (x \leq 3) \rightarrow \perp); E(y := y \times 2)(w''_2, w''_3) \\
& = \perp + (w''_2 \vDash_{\Gamma(\mathbf{A})} (x \leq 3) \rightarrow \perp); E(y := y \times 2)(w''_2, w''_3)
\end{aligned}$$

which can be, once again, instantiated for the three complete action lattices under consideration, yielding

2:

$$\begin{aligned}
- & (w'_2 \vDash_{\Gamma(\mathbf{A})} x \leq 3); E(x := x + 1)(w'_2, w'_3) + \perp = (\top \wedge \top) \vee \perp = \top \\
- & \perp + (w''_2 \vDash_{\Gamma(\mathbf{A})} (x \leq 3) \rightarrow \perp); E(y := y \times 2)(w''_2, w''_3) = \perp \vee (\top \wedge \top) = \top
\end{aligned}$$

This interpretation coincides, as expected, with the standard **if-then-else** statement.

G:

$$\begin{aligned}
- & (w'_2 \vDash_{\Gamma(\mathbf{A})} x \leq 3); E(x := x + 1)(w'_2, w'_3) + \perp = \max\{\min\{0.1, 0.7\}, 0\} = 0.1 \\
- & \perp + (w''_2 \vDash_{\Gamma(\mathbf{A})} (x \leq 3) \rightarrow \perp); E(y := y \times 2)(w''_2, w''_3) = \max\{0, \min\{0.1 \rightarrow 0, 0.9\}\} = 0
\end{aligned}$$

which expresses the weighted choice of executing $x := x + 1$ or $y := y \times 2$.

R:

$$\begin{aligned}
- & (w'_2 \vDash_{\Gamma(\mathbf{A})} x \leq 3); E(x := x + 1)(w'_2, w'_3) + \perp = \min\{1 + 7, +\infty\} = 8 \\
- & \perp + (w''_2 \vDash_{\Gamma(\mathbf{A})} (x \leq 3) \rightarrow \perp); E(y := y \times 2)(w''_2, w''_3) = \min\{+\infty, (1 \rightarrow +\infty) + 9\} = +\infty
\end{aligned}$$

giving the energy consumed by executing the alternatives $x := x + 1$ and $y := y \times 2$, weighted by the energy consumed by evaluating the predicates $(x \leq 3)$ and $(x \leq 3) \rightarrow \perp$. The value $+\infty$ corresponds to the worst case scenario of energy consumption, let us say, a value above a certain threshold, sufficiently high to be considered not reasonable for consideration in the particular application scenario. We can thus assume, in this context, that the execution of assignment $x := x + 1$ represents the best alternative in terms of energy consumption.

Lemma 5.2.1. *Let \mathbf{A} be a complete \mathbb{H} -action lattice. The following are valid formulæ in $\Gamma(\mathbf{A})$:*

$$(14.) \langle \rho_1? \rangle \rho_2 \leftrightarrow (\rho_1 \wedge \rho_2)$$

$$(15.) [\rho_1?] \rho_2 \leftrightarrow (\rho_1 \rightarrow \rho_2)$$

Proof. **(14.):**

$$\begin{aligned}
& (w \vDash_{\Gamma(\mathbf{A})} \langle \rho_1? \rangle \rho_2) \\
= & \quad \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
& \sum_{w' \in W} (\llbracket \rho_1? \rrbracket(w, w'); (w' \vDash_{\Gamma(\mathbf{A})} \rho_2)) \\
= & \quad \{ w \neq w' \Rightarrow \llbracket \rho_1? \rrbracket(w, w') = \perp, (40) \text{ and } (12) \} \\
& (w \vDash_{\Gamma(\mathbf{A})} \rho_1); (w \vDash_{\Gamma(\mathbf{A})} \rho_2) \\
= & \quad \{ ; = \cdot \text{ by } \mathbb{H} \} \\
& (w \vDash_{\Gamma(\mathbf{A})} \rho_1) \cdot (w \vDash_{\Gamma(\mathbf{A})} \rho_2) \\
= & \quad \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
& (w \vDash_{\Gamma(\mathbf{A})} \rho_1 \wedge \rho_2)
\end{aligned}$$

(15.):

$$\begin{aligned}
& (w \vDash_{\Gamma(\mathbf{A})} [\rho_1?] \rho_2) \\
= & \quad \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
& \bigwedge_{w' \in W} (\llbracket \rho_1? \rrbracket(w, w') \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho_2)) \\
= & \quad \{ \text{definition of } \llbracket \rho_1? \rrbracket + (19) \} \\
& \bigwedge_{w' \in W - \{w\}} (\perp \rightarrow (w' \vDash_{\Gamma(\mathbf{A})} \rho_2)) \cdot ((w \vDash_{\Gamma(\mathbf{A})} \rho_1) \rightarrow (w \vDash_{\Gamma(\mathbf{A})} \rho_2)) \\
= & \quad \{ \text{by Lemma 5.1.2} \} \\
& \bigwedge_{w' \in W - \{w\}} \{\top\} \cdot ((w \vDash_{\Gamma(\mathbf{A})} \rho_1) \rightarrow (w \vDash_{\Gamma(\mathbf{A})} \rho_2))
\end{aligned}$$

$$\begin{aligned}
&= \{ a \leq \top \Leftrightarrow a \cdot \top = a \} \\
&\quad (w \vDash_{\Gamma(\mathbf{A})} \rho_1) \rightarrow (w \vDash_{\Gamma(\mathbf{A})} \rho_2) \\
&= \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
&\quad (w \vDash_{\Gamma(\mathbf{A})} \rho_1 \rightarrow \rho_2)
\end{aligned}$$

□

5.3 An illustration

We explore in this section the application of the logic to a simple program. Concretely, we give examples of computing the truth degree of some logical properties, defined over programs (87) and (59).

Consider first the following example. We want to evaluate the predicate $x \geq 4$ after the execution of program (87), which can be represented by the $\Gamma(\mathbf{A})$ formula

$$\langle \langle (x \geq 3)?; x := x + 1 \rangle + \langle \langle (x \leq 3) \rightarrow \perp \rangle?; y := y \times 2 \rangle \rangle (x \geq 4)$$

Resorting on the axiomatic system given above, such calculation is obtained as follows

$$\begin{aligned}
&w_2 \vDash_{\Gamma(\mathbf{A})} \langle (x \leq 3)?; (x := x + 1) \rangle + \langle \langle (x \leq 3) \rightarrow \perp \rangle?; y := y \times 2 \rangle \rangle (x \geq 4) \\
&= \{ 3. \text{ and } 4. \text{ of lemma } 5.1.4 \} \\
&w_2 \vDash_{\Gamma(\mathbf{A})} \langle (x \leq 3)? \rangle \langle x := x + 1 \rangle (x \leq 4) \vee \langle \langle (x \leq 3) \rightarrow \perp \rangle? \rangle \langle y := y \times 2 \rangle \rangle (x \geq 4) \\
&= \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
&\llbracket (x \leq 3)? \rrbracket (w_2, w'_2); (\llbracket x := x + 1 \rrbracket (w'_2, w'_3); w'_3 \vDash_{\Gamma(\mathbf{A})} (x \geq 4)) \\
&\quad + \llbracket (x \leq 3)? \rrbracket (w_2, w''_2); (\llbracket x := x + 1 \rrbracket (w''_2, w''_3); w''_3 \vDash_{\Gamma(\mathbf{A})} (x \geq 4)) \\
&\quad + \llbracket \langle (x \leq 3) \rightarrow \perp \rangle? \rrbracket (w_2, w'_2); (\llbracket y := y \times 2 \rrbracket (w'_2, w'_3); w'_3 \vDash_{\Gamma(\mathbf{A})} (x \geq 4)) \\
&\quad + \llbracket \langle (x \leq 3) \rightarrow \perp \rangle? \rrbracket (w_2, w''_2); (\llbracket y := y \times 2 \rrbracket (w''_2, w''_3); w''_3 \vDash_{\Gamma(\mathbf{A})} (x \geq 4))
\end{aligned}$$

Concretely, we assume instantiations with the usual three action lattices, yielding

$$\mathbf{2} : [\top \wedge (\top \wedge \top) \vee \perp \wedge (\perp \wedge \top)] \wedge [\perp \wedge (\perp \wedge \top) \vee \top \wedge (\top \wedge \top)] = \top \vee \top = \top$$

This instantiation coincides with the classical case, i.e. the Boolean evaluation of logic formulas.

G :

$$\begin{aligned}
&[0.1; (0.7; 0.4) + 0; (0; 0.4)] + [0; (0; 0.4) + 0; (0.9; 0.4)] \\
&= \max\{\max\{\min\{0.1, \min\{0.7, 0.4\}\}, \\
&\quad \min\{0, \min\{0, 0.4\}\}\}, \max\{\min\{0, \min\{0, 0.4\}\}, \min\{0, \min\{0.9, 0.4\}\}\}\} \\
&= 0.1
\end{aligned}$$

The Gödel lattice gives the certainty degree of the formula.

R :

$$\begin{aligned} & [1; (7; 1) + (+\infty); (+\infty + 4)] + [+ \infty; (+\infty; 1) + 0; (9; 4)] \\ &= \min\{\min\{9, +\infty\}, \min\{+\infty, 13\}\} \\ &= 9 \end{aligned}$$

Note the singularity of this interpretation. Instead of verifying a formula, as in the Boolean case, or computing its certainty, as occurs for the Gödel algebra, this value is the energy consumed by performing the operation itself.

Let us now revisit program (59) and assume, by considering an instantiation with lattice **G**, that $E(x := x + 1)(w'_2, w'_3) = 0.6$ and $E(y := y \times 2)(w''_2, w''_3) = 0.4$. As discussed before, by perceiving (59) as a probabilistic choice, these weights are interpreted as probabilities of execution of each assignment.

We also consider, for this interpretation, the following characterisation of program states:

$$\begin{aligned} w_0(x)(r) &= \begin{cases} 1, & \text{if } r = 1 \\ 0 & \text{otherwise} \end{cases} & w_0(y)(r) &= \begin{cases} 1 & \text{if } r = 2 \\ 0, & \text{otherwise} \end{cases} \\ w_1(x)(r) &= \begin{cases} 1 & \text{if } r = 2 \\ 0, & \text{otherwise} \end{cases} & w_1(y)(r) &= \begin{cases} 1 & \text{if } r = 2 \\ 0, & \text{otherwise} \end{cases} \\ w_2(x)(r) &= \begin{cases} 1 & \text{if } r = 4 \\ 0, & \text{otherwise} \end{cases} & w_2(y)(r) &= \begin{cases} 1 & \text{if } r = 2 \\ 0, & \text{otherwise} \end{cases} \end{aligned}$$

and with the execution of the program, the states w'_3 and w''_3 become

$$\begin{aligned} w'_3(x)(r) &= \begin{cases} 1 & \text{if } r = 5 \\ 0, & \text{otherwise} \end{cases} & w'_3(y)(r) &= \begin{cases} 1 & \text{if } r = 2 \\ 0, & \text{otherwise} \end{cases} \\ w''_3(x)(r) &= \begin{cases} 1 & \text{if } r = 4 \\ 0, & \text{otherwise} \end{cases} & w''_3(y)(r) &= \begin{cases} 1 & \text{if } r = 4 \\ 0, & \text{otherwise} \end{cases} \end{aligned}$$

and, consequently, the evaluation of the predicate in the same states is $\llbracket x \geq 4 \rrbracket_{w'_3} = 1$ and $\llbracket x \geq 4 \rrbracket_{w''_3} = 0$.

In this particular context, the logic is capable of asking, for example, "what is the probability of reaching a state which satisfies $x \geq 4$ ", corresponding to the computation of the $\Gamma(\mathbf{G})$ formula $\langle (x := x + 1) + (y := y \times 2) \rangle (x \geq 4)$, as follows

$$\begin{aligned}
& w_2 \vDash_{\Gamma(\mathbf{A})} \langle (x := x + 1) + (y := y \times 2) \rangle (x \geq 4) \\
= & \quad \{ \text{3. of lemma 5.1.4} \} \\
& w_2 \vDash_{\Gamma(\mathbf{A})} \langle x := x + 1 \rangle (x \geq 4) \vee \langle y := y \times 2 \rangle (x \geq 4) \\
= & \quad \{ \text{definition of } \vDash_{\Gamma(\mathbf{A})} \} \\
& \llbracket x := x + 1 \rrbracket (w_2, w'_3); w'_3 \vDash_{\Gamma(\mathbf{A})} (x \geq 4) + \llbracket y := y \times 2 \rrbracket (w_2, w''_3); w''_3 \vDash_{\Gamma(\mathbf{A})} (x \geq 4) \\
= & \quad \{ \text{definition of } \mathbf{G} \} \\
& \max\{\min\{0.6, 1\}, \min\{0.4, 0\}\} = 0.6
\end{aligned}$$

The value 0.6 is the probability that the program (59) reaches a state satisfying $x \leq 4$. An analogous computation can be made if the program (59) yields a sub-probabilistic choice instead.

5.4 Bisimulation

We introduce in this section a parametric notion of bisimulation, and we prove its modal invariance for any $\Gamma(\mathbf{A})$. The bisimulation generalises the notion recently introduced in [JMM20] in the context of fuzzy modal logic. In this section we resort to a linear complete action lattice as parameter, since the linearity is necessary to establish some of the results. In order to distinguish the semantics and the satisfaction relation over different models, we assume notations $\llbracket _ \rrbracket_w^M$ and $M, w \vDash_{\Gamma(\mathbf{A})}$, for a model $M \in \text{Mod}^{\Gamma(\mathbf{A})}$ and a state $w \in W$.

Let us first recall some handy properties of action lattices from [MNM16].

$$x \leq y \Rightarrow x; a \leq y; a \tag{88}$$

$$a \leq b \ \& \ c \leq d \Rightarrow a + c \leq b + d \tag{89}$$

Definition 5.4.1 (Π -Bisimulation). *Let X be a set of variables, and $\Delta = ((F, P), \Pi)$, where (F, P) is a data signature and $\Pi \subset \text{Prog}_0$, $M = (W, V, E)$ and $M' = (W', V', E')$ be, respectively, a signature and two $\Gamma(\mathbf{A})$ -models, for any linear complete action lattice \mathbf{A} . A Π -bisimulation from M to M' is a non empty relation $B \subseteq W \times W'$ such that whenever $w B w'$, the following conditions hold:*

(ATOMS) *for any $x \in X, r \in \mathbb{R}$, $\llbracket x \rrbracket_w^M(r) = \llbracket x \rrbracket_{w'}^{M'}(r)$ and, for any $p \in T_P(X)$, $\llbracket p \rrbracket_w^M = \llbracket p \rrbracket_{w'}^{M'}$*

(FZIG) *for any $u \in W$ and $\pi \in \Pi$, $\llbracket \pi \rrbracket_0^M(w, u) \leq \sum_{u' \in B[\{u\}]} \llbracket \pi \rrbracket_0^{M'}(w', u')$*

(FZAG) *for any $u' \in W'$ and $\pi \in \Pi$, $\llbracket \pi \rrbracket_0^{M'}(w', u') \leq \sum_{u \in B^{-1}[\{u'\}]} \llbracket \pi \rrbracket_0^M(w, u)$*

We write $w \sim w'$ whenever, there is a bisimulation B such that $(w, w') \in B$.

Next result generalises the notion of Π -Bisimulation to any program in Prog .

Proposition 5.4.1. *Let \mathbf{A} be a linear complete action lattice, Δ a data signature, and $M = (W, V, E)$, $M' = (W', V', E')$ two $\Gamma(\mathbf{A})$ -models for Δ . Then, any Π -bisimulation over $\Gamma(\mathbf{A})$ -models is a Prog-bisimulation.*

Proof. The proof is done by induction over the programs structure. Let $B \subseteq W \times W'$ be a bisimulation and $w \in W, w' \in W'$ such that $(w, w') \in B$.

The result for atomic programs is given by hypothesis. Let us prove the **(Fzig)** condition for programs $\pi; \pi'$. By induction hypothesis, let us assume that **(Fzig)** of B for π and π' . Hence, for any $v \in W$

$$\llbracket \pi \rrbracket^M(w, v) \leq \sum_{v' \in B(v)} \llbracket \pi \rrbracket^{M'}(w', v') \quad (90)$$

holds. By (24), we have also that, for any $v \in W$ there is a $v'_v \in B(v)$ such that

$\sum_{v' \in B(v)} \llbracket \pi \rrbracket^{M'}(w', v') = \llbracket \pi \rrbracket^{M'}(w', v'_v)$. Moreover, since $(v, v'_v) \in B$, we have by **(Fzig)** of B for π' that

$$\llbracket \pi' \rrbracket^M(v, u) \leq \sum_{u' \in B(u)} \llbracket \pi' \rrbracket^{M'}(v'_v, u')$$

By (88) in (90) we get, for any $v \in W$,

$$\llbracket \pi \rrbracket^M(w, v); \llbracket \pi' \rrbracket^M(v, u) \leq \llbracket \pi \rrbracket^{M'}(w', v'_v); \sum_{u' \in B(u)} \llbracket \pi' \rrbracket^{M'}(v'_v, u')$$

and, by (89),

$$\sum_{v \in W} \llbracket \pi \rrbracket^M(w, v); \llbracket \pi' \rrbracket^M(v, u) \leq \sum_{v'_v \in W'} \llbracket \pi \rrbracket^{M'}(w', v'_v); \sum_{u' \in B(u)} \llbracket \pi' \rrbracket^{M'}(v'_v, u') \quad (91)$$

Moreover, since $\{v'_v : v \in W\} \subseteq \{v' : v' \in W'\}$, and by (29), (25) and (26), we have that

$$\sum_{v'_v \in W'} \llbracket \pi \rrbracket^{M'}(w', v'_v); \sum_{u' \in B(u)} \llbracket \pi' \rrbracket^{M'}(v'_v, u') \leq \sum_{u' \in B(u)} \left(\sum_{v' \in W'} (\llbracket \pi \rrbracket^{M'}(w', v'); \llbracket \pi' \rrbracket^{M'}(v', u')) \right) \quad (92)$$

By (91) and (92), we achieve $\llbracket \pi; \pi' \rrbracket^M(w, u) \leq \sum_{u' \in B(u)} \llbracket \pi; \pi' \rrbracket^{M'}(w', u')$. The proof of **(Fzag)** condition is analogous.

For a program $\pi + \pi'$, we observe that

$$\begin{aligned} & \llbracket \pi + \pi' \rrbracket^M(w, u) \\ = & \quad \{ \text{interpretation of programs} \} \\ & \llbracket \pi \rrbracket^M(w, u) + \llbracket \pi' \rrbracket^M(w, u) \\ \leq & \quad \{ \mathbf{(Fzig)} \text{ and (89)} \} \\ & \sum_{u' \in B(u)} \llbracket \pi \rrbracket^{M'}(w', u') + \sum_{u' \in B(u)} \llbracket \pi' \rrbracket^{M'}(w', u') \end{aligned}$$

$$= \quad \{ \text{definition of } + \}$$

$$\sum_{u' \in B(u)} \llbracket \pi + \pi' \rrbracket^{M'}(w', u')$$

Finally, for a program π^* , we observe that, by definition of $*$,

$$\llbracket \pi^* \rrbracket^M(w, u) = \sum_{k \geq 0} (\llbracket \pi \rrbracket^k)^M(w, u) = (\llbracket \pi \rrbracket^0)^M(w, u) + \llbracket \pi \rrbracket^M(w, u) + (\llbracket \pi \rrbracket^2)^M(w, u) + \dots$$

But, for each k , $(\llbracket \pi \rrbracket^k)^M(w, u) \leq \sum_{u' \in B(u)} (\llbracket \pi \rrbracket^k)^{M'}(w', u')$, by Fzig. Hence,

$$\begin{aligned} & \sum_{k \geq 0} (\llbracket \pi \rrbracket^k)^M(w, u) \\ & \leq \quad \{ (89) \} \\ & \sum_{k \geq 0} \left(\sum_{u' \in B(u)} (\llbracket \pi \rrbracket^k)^{M'}(w', u') \right) \\ & = \quad \{ (25) \text{ and } (26) \} \\ & \sum_{u' \in B(u)} \left(\sum_{k \geq 0} (\llbracket \pi \rrbracket^k)^{M'}(w', u') \right) \\ & = \quad \{ \text{definition of } * \} \\ & \sum_{u' \in B(u)} (\llbracket \pi^* \rrbracket)^{M'}(w', u') \end{aligned}$$

□

Next result establishes the well-known *word bisimulation result* on this weighted setting. This result reduces the invariance property of formulas involving compound programs in Prog to the one involving just the set of atomic programs Prog₀. In other words, it reduces the modal invariance problem of a generated dynamic logic to the modal invariance of the underlying logic.

Theorem 5.4.1 (Modal invariance). *Let Δ be a data signature, \mathbf{A} a linear complete action lattice, $M = (W, V, E)$ and $M' = (W', V', E')$ two $\Gamma(\mathbf{A})$ -models for Δ , and $B \subseteq W \times W'$ a bisimulation from M to M' . Then, for any $w \in W$, $w' \in W'$ such that $w \sim w'$ and for all formulas $\rho \in \text{Fm}^{\Gamma(\mathbf{A})}(\Delta)$,*

$$(M, w \vDash_{\Gamma(\mathbf{A})} \rho) = (M', w' \vDash_{\Gamma(\mathbf{A})} \rho)$$

Proof.

We prove this result by induction on the structure of formulas.

For the invariance of the formula \top , note that $(M, w \vDash_{\Gamma(\mathbf{A})} \top) = \top = (M', w' \vDash_{\Gamma(\mathbf{A})} \top)$ and similarly for the formula \perp .

Invariance of $p \in T_P(X)$ is a direct consequence of **(Atoms)**,

$$(M, w \vDash_{\Gamma(\mathbf{A})} p) = \llbracket p \rrbracket_w^M = \llbracket p \rrbracket_{w'}^{M'} = (M', w' \vDash_{\Gamma(\mathbf{A})} p)$$

For the invariance of formulas $\varphi \wedge \psi$, we observe that

$$(M, w \vDash_{\Gamma(\mathbf{A})} \rho_1 \wedge \rho_2) = (M, w \vDash_{\Gamma(\mathbf{A})} \rho_1) \cdot (M, w \vDash_{\Gamma(\mathbf{A})} \rho_2) \stackrel{I.H.}{=}.$$

$$(M', w' \vDash_{\Gamma(\mathbf{A})} \rho_1) \cdot (M', w' \vDash_{\Gamma(\mathbf{A})} \rho_2) = (M', w' \vDash_{\Gamma(\mathbf{A})} \rho_1 \wedge \rho_2)$$

The proofs for the invariance of formulas $\rho_1 \vee \rho_2$ and $\rho_1 \rightarrow \rho_2$ are similar.

Now it just remains to prove formulas $\langle \pi \rangle \rho$ and $[\pi] \rho$. Since \mathbf{A} is linear, we have, by Proposition 5.4.1 that it is enough to prove the invariance for formulas involving atomic programs $\pi_0 \in \text{Prog}_0$. For the invariance of a formula $\langle \pi_0 \rangle \rho$, we observe that, by **(Fzig)**, we have

$$\forall_{u \in W}, \llbracket \pi_0 \rrbracket_0^M(w, u) \leq \sum_{u' \in E[\{u\}]} \llbracket \pi_0 \rrbracket_0^{M'}(w', u') = \llbracket \pi_0 \rrbracket_0^{M'}(w', u'_u), \text{ for some } u'_u \in W' \quad (93)$$

Since $\forall_{u \in W}, u'_u \in B[\{u\}]$, we have $u B u'_u$. By I. H., we have $(M, u \vDash_{\Gamma(\mathbf{A})} \rho) = (M', u'_u \vDash_{\Gamma(\mathbf{A})} \rho)$ and, by (93),

$$\forall_{u \in W}, \llbracket \pi_0 \rrbracket_0^M(w, u) \cdot (M, u \vDash_{\Gamma(\mathbf{A})} \rho) \leq \llbracket \pi_0 \rrbracket_0^{M'}(w', u'_u) \cdot (M', u'_u \vDash_{\Gamma(\mathbf{A})} \rho)$$

In particular,

$$\sum_{u \in W} (\llbracket \pi_0 \rrbracket_0^M(w, u) \cdot (M, u \vDash_{\Gamma(\mathbf{A})} \rho)) \leq \sum_{u'_u \in W'} (\llbracket \pi_0 \rrbracket_0^{M'}(w', u'_u) \cdot (M', u'_u \vDash_{\Gamma(\mathbf{A})} \rho)) \quad (94)$$

Since $\{u'_u : u \in W\} \subseteq \{u' : u' \in W'\}$ we have $\sum \{u'_u : u \in W\} \leq \sum \{u' : u' \in W'\}$ and, by (94),

$$\sum_{u \in W} (\llbracket \pi_0 \rrbracket_0^M(w, u) \cdot (M, u \vDash_{\Gamma(\mathbf{A})} \rho)) \leq \sum_{u' \in W'} (\llbracket \pi_0 \rrbracket_0^{M'}(w', u') \cdot (M', u' \vDash_{\Gamma(\mathbf{A})} \rho))$$

i.e. $(M, w \vDash_{\Gamma(\mathbf{A})} \langle \pi_0 \rangle \rho) \leq (M', w' \vDash_{\Gamma(\mathbf{A})} \langle \pi_0 \rangle \rho)$. Similarly we can prove $(M, w \vDash_{\Gamma(\mathbf{A})} \langle \pi_0 \rangle \rho) \geq (M', w' \vDash_{\Gamma(\mathbf{A})} \langle \pi_0 \rangle \rho)$ by using **(Fzag)** condition.

For the invariance of formulas $[\pi_0] \rho$, with $\pi_0 \in \text{Prog}_0$, since $w E w'$ we have, by **(Fzig)**,

$$\forall_{u \in W}, \llbracket \pi_0 \rrbracket_0^M(w, u) \leq \sum_{u' \in E[\{u\}]} \llbracket \pi_0 \rrbracket_0^{M'}(w', u') = \llbracket \pi_0 \rrbracket_0^{M'}(w', u'_u), \text{ for some } u'_u \in W' \quad (95)$$

Since $\forall_{u \in W}, u'_u \in B[\{u\}]$, we have $u \in W, u B u'_u$. Hence, by I.H.

$$(M, u \vDash_{\Gamma(\mathbf{A})} \rho) = (M', u'_u \vDash_{\Gamma(\mathbf{A})} \rho) \quad (96)$$

It follows from the definition of I that $x_0 \leq x_1$ implies $I(x_0, y) \geq I(x_1, y)$. Then, from (95) and (96), we have

$$\forall_{u \in W}, I(\llbracket \pi_0 \rrbracket_0^M(w, u), (M, u \vDash_{\Gamma(\mathbf{A})} \rho)) \geq I(\llbracket \pi_0 \rrbracket_0^{M'}(w', u'_u), (M', u'_u \vDash_{\Gamma(\mathbf{A})} \rho))$$

and, in particular,

$$\bigwedge_{u \in W} (I(\llbracket \pi_0 \rrbracket_0^M(w, u), (M, u \vDash_{\Gamma(\mathbf{A})} \rho))) \geq \bigwedge_{u'_u : u \in W} (I(\llbracket \pi_0 \rrbracket_0^{M'}(w', u'_u), (M', u'_u \vDash_{\Gamma(\mathbf{A})} \rho))) \quad (97)$$

Since $\{u'_u : u \in W\} \subseteq \{u' : u' \in W'\}$, we have $\bigwedge \{u'_u : u \in W\} \geq \bigwedge \{u' : u' \in W'\}$. Hence

$$\bigwedge_{u \in W} (I(\llbracket \pi_0 \rrbracket_0^M(w, u), (M, u \vDash_{\Gamma(\mathbf{A})} \rho))) \geq \bigwedge_{u' \in W'} (I(\llbracket \pi_0 \rrbracket_0^{M'}(w', u'), (M', u' \vDash_{\Gamma(\mathbf{A})} \rho))) \quad (98)$$

and therefore $(M, w \vDash_{\Gamma(\mathbf{A})} [\pi_0] \rho) \geq (M', w' \vDash_{\Gamma(\mathbf{A})} [\pi_0] \rho)$. The proof for $(M, w \vDash_{\Gamma(\mathbf{A})} [\pi_0] \rho) \leq (M', w' \vDash_{\Gamma(\mathbf{A})} [\pi_0] \rho)$ is analogous. \square

Part II

WEIGHTED *MULTI-FLOW* COMPUTATIONS

Context

A specific type of system where the notion of weight plays a major role is *fuzzy control systems* [Zad65]. Built on the rules of fuzzy logic, they emerged essentially as a formalisation of knowledge for expressing properties that cannot be evaluated in simple terms of “true” and “false”.

Programs for those systems adopt a distinct interpretation of conditional statements from classic imperative programming languages. What we mean is that their execution is not nondeterministic, adopting instead a weighted parallel behaviour, as shown in Example 1.3.2. Programs for fuzzy control systems are implemented using *fuzzy control languages*, with applications e.g. in medical diagnosis and robotics. One example of the former is the *Fuzzy Arden Syntax (FAS)* [VMA10], an extension to *Arden Syntax (AS)* [Hri94] to cater for vague or uncertain information often arising in clinical situations. One example of the later is *jFuzzyLogic*, implemented in [CA12] in one case study to analyse the behaviour of a robot in a labyrinth. Due to their intuitiveness, being very close to natural language, FAS and jFuzzyLogic are used to design knowledge-based components in medical decision support systems [ACB⁺18, SFdBA12, SHJ⁺94], and in artificial intelligence [CAf13], respectively.

Thus rigorous semantics structures and logics for reasoning about this class of programs are highly required. The concept of *binary multirelation*, formalised as a (crisp) relation between a state and a set of states, points towards this direction [Rew03]. In this reference, authors explore more deeply these relations, by doing an extensive algebraic characterisation, as well as defining composition, union and intersection operators over them. Although it appeared initially in the context of game logics, the concept was used to give semantics to *concurrent propositional dynamic logic (CPDL)*, an extension to PDL to reason about programs running in parallel [Pel87].

Overview

Dynamic logics such as CPDL are intended to capture programs with a parallel behaviour, but still not able to capture weighted “multi-flow” computations, as arising in FAS. In fact, there is an absence of rigorous ways for reasoning about such class of programs. In this direction, we contribute, in **Part 2** of this thesis, with semantic structures (algebras and logics) for \mathcal{F}_2 -programs interpreted as “multi-flow” computations, endowed with a family of dynamic logics for their formal verification.

First, in order to build these semantic constructions, we model the most basic notion of a program as a *weighted binary multirelation*. Such a concept represents a generalisation of a binary multirelation [Rew03], to model conditionals as parallel executions of two or more actions, each one endowed with a (possibly different) weight. Taking this notion as the basic construction, we define an algebra of programs and prove that it defines a *proto-trioid*, i.e. a variant of a semiring where *parallel composition* and *union* are associative whereas the *sequential composition* is not, and where the later also does not left distribute over union (both axioms are inequalities). The *parallel composition* is then taken as the core ingredient to recursively define arbitrary (compound) programs as weighted “multi-flow” computations. A discussion about the differences in the axiomatisation, when comparing with an idempotent semiring and a Kleene algebra,

is also conducted. In particular we provide counter-examples showing that associativity of sequential composition and its left distribution over union are not indeed equalities.

Second, the relational semantics is built analogously to what was done in **Part 1**, taking computational states as weighted valuations of variables over a given domain, and programs as their modifiers. A compound program is, as explained above, modelled as a weighted multirelation built from operations of the defined program algebra.

Third, a family of dynamic logics is generated, parametric on a complete right residuated lattice, which, depending on each instantiation, gives different meanings to the notion of a computation. To give semantics to **if-then-else** and **switch** instructions, a proper notion of test is introduced. The ‘aggregation’ and ‘defuzzify’ operators, present in the syntax of fuzzy programming languages, are also captured using the proposed semantics, to combine the multiple branches resulting from the parallel execution of the conditionals. A set of dynamic logic properties are proved for the generated logics, which allow compositional reasoning over \mathcal{F}_2 -programs.

This framework is illustrated by two examples in which some properties of simple fuzzy programs are proved: one using fuzzy Arden syntax, the other written in jFuzzyLogic.

Roadmap

Part 2 is organised as follows. Chapter 6 presents the algebraic construction to capture \mathcal{F}_2 -programs. First, Section 6.1 recaps the concept of binary multirelation. Then, Section 6.2 introduces weighted binary multirelation, which, together with proper generalisations of sequential and parallel compositions of binary multirelations, form the program algebra to model weighted “multi-flow” computations. Based on such a structure, a semantics for \mathcal{F}_2 -programs is presented including, naturally, program variables and assignments, in Chapter 7. Chapter 8 introduces a family of dynamic logics for the verification of \mathcal{F}_2 -programs. Finally, the framework is illustrated by proving some properties of two simple programs: the first (Section 8.3), written in FAS, revisits Example 1.3.2; the second (Section 5.3) corresponds to a fuzzy control system for controlling a container crane, written in JFuzzyLogic.

ALGEBRAS OF WEIGHTED *MULTI-FLOW* COMPUTATIONS

We follow the same steps as in **Part 1** by presenting, first, in this chapter, an algebra to interpret programs. We resort to the theory of binary multirelations, generalised to capture fuzziness, to develop such an algebraic structure. As enhanced in the introduction, the concept of binary multirelation may act as an input-output semantics for describing the execution of a program from a single input state to a set of output states. The generalisation to the weighted case goes even further, and gives us the framework that we need to capture the computations designated by weighted “multi-flow”. Additionally, by defining some operators, we intend to define a suitable algebra of programs to collect these computations. Unlike the structure obtained in **Part 1**, however, such an algebra is not a Kleene algebra. This limitation is in part, as we will see, due to the definition of sequential composition which, due to the nature of the mathematical object over which they operate, is not associative and does not left distribute over union.

6.1 Preliminaries: binary multirelations

Definition 6.1.1 (Binary multirelation [Rew03]). *Let W be a set. A binary multirelation is a subset of the cartesian product $W \times \mathbf{2}^W$, i.e. a set of ordered pairs (u, U) , where $u \in W$ and $U \subseteq W$.*

The space of binary multirelations over W is denoted $M(W)$.

Given multirelations $R, S \in M(W)$, the *union* of R and S , $R \cup S$, is simply the set union, and the *parallel composition* [FS16] is given by

$$R||S = \{(w, U \cup V) \mid (w, U) \in R \wedge (w, V) \in S\} \quad (99)$$

The operator indicates a parallel run of a program leading from a state w to a set of states in $U \cup V \subseteq W$, “combining” the arriving states of R and S into W .

To illustrate better the differences between operators \cup and $||$, let us consider the following example.

Example 6.1.1. *Consider the multirelations $R = \{(w_0, \{u_1, u_2\})\}$ and $S = \{(w_0, \{v_1\})\}$. The union of R and S is the set $\{(w_0, \{u_1, u_2\}), (w_0, \{v_1\})\}$, which carries the standard interpretation of conditionals as nondeterministic choice between $\{w_0, \{u_1, u_2\}\}$ and $\{(w_0, \{v_1\})\}$. On the other hand, the parallel composition $R||S$ represents a single execution $\{(w_0, \{u_1, u_2, v_1\})\}$ from state w_0 going simultaneously to states u_1, u_2 and v_1 .*

As mentioned in the introduction, the literature presents three distinct sequential composition operators for binary multirelations: Kleisli, Peleg and Parikh. The former is the relational generalisation of the standard composition on the Kleisli category of the powerset monad [Mac71]. The Peleg composition is used in the semantics of CPDL [Pel87], and thus its motivations arise from program logics. The Parikh sequential composition, on the other hand, has very distinct motivations, being introduced for the first time in [Par83] in the context of game logics. For this reason we left the discussion of the latter for the future, presenting instead, in this section, the definitions of Kleisli and Peleg compositions for the classic case, and in the next section, their generalisations for the weighted case, accompanied by a simple example.

Definition 6.1.2 (Kleisli sequential composition). *Let W be a set. Consider two binary multirelations $R, S \in M(W)$. The Kleisli sequential composition of R and S is defined as*

$$R \circledast S = \{(w, U) \mid \exists V \cdot (w, V) \in R \wedge U = \bigcup S(V)\}$$

A pair (w, U) is in $R \circledast S$ if w is related by R with some intermediate set V and U is the union of all the images of V by S .

To illustrate the operator, let us consider the binary multirelations $R = \{(a, \{a, b\}), (a, \{a\}), (b, \{a\})\}$ and $S = \{(a, \{a\}), (a, \{b\})\}$. Their Kleisli composition $R \circledast S$ is computed as

$$\begin{aligned} (a, \{a, b\}) \in R \circledast S &\Leftrightarrow \\ \exists \{a\} \cdot (a, \{a\}) \in R \wedge \{a, b\} &= \bigcup S(\{a\}) \end{aligned}$$

But $\bigcup S(\{a\}) = S(a) = \{a\} \cup \{b\} = \{a, b\}$. Hence $(a, \{a, b\}) \in R \circledast S$.

$$\begin{aligned} (b, \{a, b\}) \in R \circledast S &\Leftrightarrow \\ \exists \{a\} \cdot (b, \{a\}) \in R \wedge \{a, b\} &= \bigcup S(\{a\}) \end{aligned}$$

And $\bigcup S(\{a\}) = S(a) = \{a\} \cup \{b\} = \{a, b\}$. Therefore $(a, \{a, b\}) \in R \circledast S$. On the other hand,

$$\begin{aligned} (a, \{a\}) \in R \circledast S &\Leftrightarrow \\ \exists \{a, b\} \cdot (b, \{a\}) \in R \wedge \{a\} &= \bigcup S(\{a, b\}) \end{aligned}$$

But $\bigcup S(\{a, b\}) = S(a) \cup S(b) = \{a\} \cup \{b\} \cup \{\} = \{a, b\} \neq \{a\}$. Hence $(a, \{a\}) \notin R \circledast S$, and analogously, we can observe that $(a, \{b\}), (b, \{a\}), (b, \{b\}) \notin R \circledast S$.

The Kleisli composition is the multirelation $R \circledast S = \{(a, \{a, b\}), (b, \{a, b\})\}$.

Definition 6.1.3 (Peleg sequential composition [FS16]). Consider two binary multirelations $R, S \in M(W)$. The Peleg sequential composition of R and S is given by

$$R \circ S = \{(w, U) \mid \exists_V \cdot (w, V) \in R \wedge \exists_{F:V \rightarrow W} \cdot (\forall_{v \in V} \cdot (v, F(v)) \in S) \wedge U = \bigcup F(V)\}$$

A pair (w, U) is in $R \circ S$ if w is related by R with some intermediate set V and each element in V is related by S to a set $F(V)$ such that $U = \bigcup F(V)$.

Let us consider, again, the multirelations $R = \{(a, \{a, b\}), (a, \{a\}), (b, \{a\})\}$ and $S = \{(a, \{a\}), (a, \{b\})\}$. The Peleg composition of R and S is computed the following way.

$$\begin{aligned} (a, \{a\}) \in R \circ S &\Leftrightarrow \\ \exists \{a, b\} \cdot (a, \{a, b\}) \in R &\wedge \exists_{F:V \rightarrow W} \cdot (a, F(a)) \in S \wedge \{a\} = \bigcup F(\{a\}) \end{aligned}$$

making F such that $F(a) = \{a\}$. Therefore, $(a, \{a\}) \in R \circ S$.

$$\begin{aligned} (a, \{b\}) \in R \circ S &\Leftrightarrow \\ \exists \{a\} \cdot (a, \{a\}) \in R &\wedge \exists_{F:V \rightarrow W} \cdot (a, F(a)) \in S \wedge \{b\} = \bigcup F(\{a\}) \end{aligned}$$

with F such that $F(a) = \{b\}$. Thus, $(a, \{b\}) \in R \circ S$.

$$\begin{aligned} (b, \{a\}) \in R \circ S &\Leftrightarrow \\ \exists \{a\} \cdot (b, \{a\}) \in R &\wedge \exists_{F:V \rightarrow W} \cdot (a, F(a)) \in S \wedge \{a\} = \bigcup F(\{a\}) \end{aligned}$$

with F such that $F(a) = \{a\}$. Hence, $(b, \{a\}) \in R \circ S$.

$$\begin{aligned} (b, \{b\}) \in R \circ S &\Leftrightarrow \\ \exists \{a\} \cdot (b, \{a\}) \in R &\wedge \exists_{F:V \rightarrow W} \cdot (a, F(a)) \in S \wedge \{b\} = \bigcup F(\{a\}) \end{aligned}$$

with F such that $F(a) = \{b\}$. We conclude $(b, \{b\}) \in R \circ S$.

Hence we obtain the composition $R \circ S = \{(a, \{a\}), (a, \{b\}), (b, \{a\}), (b, \{b\})\}$, and we conclude that

$$R \circ S = \{(a, \{a, b\}), (b, \{a, b\})\} \neq R \circ S = \{(a, \{a\}), (a, \{b\}), (b, \{a\}), (b, \{b\})\}$$

As described in the Introduction, the semantics of \mathcal{F}_2 -programs is based on the concept of weighted multirelation, which we introduce below.

6.2 Introducing weighted multirelations

Definition 6.2.1 (Weighted binary multirelation). *Let W be a set and \mathbf{L} a complete right residuated lattice. A weighted binary multirelation is a set $R \subseteq W \times \mathbf{L}^W$.*

We denote by $M^{\mathbf{L}}(W)$ the space of \mathbf{L} -valued weighted binary multirelations over a set W .

Note, particularly, how this definition generalises the concept of binary multirelation (Definition 6.1.1), replacing \mathbf{L} by $\mathbf{2}$, thus supporting a wider range of truth values. Therefore, a program modelled as a weighted multirelation represents an execution with multiple “arrows” leaving a state into a set of states in parallel, with (possible different) weights associated with each “arrow”. Note also that if \mathbf{L} is the Boolean lattice $\mathbf{2}$, any weighted binary multirelation $R \subseteq W \times \mathbf{2}^W$ is a binary multirelation. Since the goal of this chapter is still to model programs as binary input-output relations, only the binary case is considered, i.e. weighted multirelations defined as $R \subseteq W \times \mathbf{L}^W$, and thus we always refer to weighted binary multirelations simply as weighted multirelations.

We now define the operators of *union*, *parallel* and *sequential* compositions for weighted multirelations, as generalisations of the correspondent classical definitions. Given weighted multirelations R and S , their *union* $R \cup S$ is just the set union. Their *parallel composition* is given by

$$R \uplus S = \{(w, \varphi_R \cup \varphi_S) \mid (w, \varphi_R) \in R \wedge (w, \varphi_S) \in S\} \quad (100)$$

where $(\varphi_R \cup \varphi_S)(w) = \varphi_R(w) + \varphi_S(w)$, for each $w \in W$, with symbol $+$ on the right hand side being the sum on lattice \mathbf{L} (as in Definition 3.4.1). The resulting multirelation $R \uplus S$ collects pairs from R and S whose first component is the same in both relations. Note that this definition generalises the one given above for binary multirelations.

Definition 6.2.2 (Kleisli sequential composition for the weighted case). *Let us consider two weighted multirelations $R, S \in M^{\mathbf{L}}(W)$. The Kleisli sequential composition of R and S is defined as*

$$R \circledast S = \{(w, \varphi) \mid \exists \varphi' \cdot (w, \varphi') \in R \wedge \varphi(u) = \sum_{(v, \psi) \in S} (\varphi'(v); \psi(u))\} \quad (101)$$

To illustrate the operator, let us consider the weighted multirelations

$R = \{(a, \varphi_1), (a, 1_\circ), (b, \varphi_2)\}$, where $\varphi_1(a) = \alpha$, $\varphi_1(b) = \beta$, $\varphi_2(a) = \gamma$ and 0 otherwise, and the weighted multirelation $S = \{(a, 1_\circ), (a, \varphi_3)\}$, where $\varphi_3(b) = \delta$ and 0 otherwise, with $\alpha, \beta, \gamma, \delta \in \mathbf{L}$.

The Kleisli composition $R \circledast S$ results in

$$R \circledast S = \{(a, \phi), (a, \phi'), (a, \rho), (a, \rho'), (b, \sigma), (b, \theta), (a, \omega), (a, \omega'), (b, \tau)\}$$

with

$$\begin{aligned}
(a, \phi) \in R \ ; \ S &\Leftrightarrow \\
\exists_{1_o} \cdot (a, 1_o) \in R \ \wedge \ \phi(a) &= 1_o(a); S(a)(a) \\
&= 1_o(a); 1_o(a) \\
&= 1; 1 \\
&= 1
\end{aligned}$$

$$\begin{aligned}
(a, \phi') \in R \ ; \ S &\Leftrightarrow \\
\exists_{\varphi_1} \cdot (a, \varphi_1) \in R \ \wedge \ \phi'(a) &= \varphi_1(a); S(a)(a) + \varphi_1(b); S(b)(a) \\
&= \varphi_1(a); 1_o(a) + \varphi_1(b); 0 \\
&= \alpha
\end{aligned}$$

$$\begin{aligned}
(a, \rho) \in R \ ; \ S &\Leftrightarrow \\
\exists_{1_o} \cdot (a, 1_o) \in R \ \wedge \ \rho(b) &= 1_o(a); S(a)(b) \\
&= 1_o(a); \varphi_3(b) \\
&= 1; \delta \\
&= \delta
\end{aligned}$$

$$\begin{aligned}
(a, \rho') \in R \ ; \ S &\Leftrightarrow \\
\exists_{\varphi_1} \cdot (a, \varphi_1) \in R \ \wedge \ \rho'(b) &= \varphi_1(a); S(a)(b) + \varphi_1(b); S(b)(b) \\
&= \varphi_1(a); \varphi_3(b) + \varphi_1(b); 0 \\
&= \alpha; \delta
\end{aligned}$$

$$\begin{aligned}
(b, \sigma) \in R \ ; \ S &\Leftrightarrow \\
\exists_{\varphi_2} \cdot (b, \varphi_2) \in R \ \wedge \ \sigma(a) &= \varphi_2(a); S(a)(a) \\
&= \varphi_2(a); 1_o(a) \\
&= \gamma; 1 \\
&= 1
\end{aligned}$$

$$\begin{aligned}
(b, \theta) \in R \circledast S &\Leftrightarrow \\
\exists_{\varphi_2} \cdot (b, \varphi_2) \in R \wedge \theta(b) &= \varphi_2(a); S(a)(b) \\
&= \varphi_2(a); \varphi_3(b) \\
&= \gamma; \delta
\end{aligned}$$

$$\begin{aligned}
(a, \omega) \in R \circledast S &\Leftrightarrow \\
\exists_{\varphi_1} \cdot (a, \varphi_1) \in R \wedge \omega(a) &= \varphi_1(a); S(a)(a) + \varphi_1(b); S(b)(a) \\
&= \varphi_1(a); 1_{\circ}(a) + \varphi_1(b); 0 \\
&= \alpha; 1 \\
&= \alpha \\
\wedge \omega(b) &= \varphi_1(a); S(a)(b) + \varphi_1(b); S(b)(b) \\
&= \varphi_1(a); \varphi_3(b) + \varphi_1(b); 0 \\
&= \alpha; \delta
\end{aligned}$$

$$\begin{aligned}
(a, \omega') \in R \circledast S &\Leftrightarrow \\
\exists_{1_{\circ}} \cdot (a, 1_{\circ}) \in R \wedge \omega'(a) &= 1_{\circ}(a); S(a)(a) \\
&= 1_{\circ}(a); 1_{\circ}(a) \\
&= 1 \\
\wedge \omega'(b) &= 1_{\circ}(a); S(a)(b) \\
&= 1_{\circ}(a); \varphi_3(b) \\
&= 1; \delta = \delta
\end{aligned}$$

$$\begin{aligned}
(b, \tau) \in R \circledast S &\Leftrightarrow \\
\exists_{\varphi_2} \cdot (b, \varphi_2) \in R \wedge \tau(a) &= \varphi_2(a); S(a)(a) \\
&= \varphi_2(a); 1_{\circ}(a) \\
&= \gamma; 1 \\
&= \gamma \\
\wedge \tau(b) &= \varphi_2(a); S(a)(b)
\end{aligned}$$

$$\begin{aligned}
&= \varphi_2(a); \varphi_3(b) \\
&= \gamma; \delta
\end{aligned}$$

Definition 6.2.3 (Peleg sequential composition for the weighted case). *Let us consider two weighted multirelations R and S over W . The Peleg sequential composition of R and S is defined as*

$$\begin{aligned}
R \circ S = \{ (w, \varphi) \mid \exists_{\varphi'} (w, \varphi') \in R \wedge (\exists_{F:W \rightarrow \mathbf{L}^W} \forall_{v \in W} (v, F(v)) \in S) \\
\wedge \varphi(u) = \sum_{v \in W} (\varphi'(v); F(v)(u)) \} \quad (102)
\end{aligned}$$

This definition is close to the one presented for *probabilistic multirelations* [Tsu12], however, with distinct motivations. In that case, the notion of multirelation is adapted to give semantics to probabilistic programs, namely to cater for the representation of probabilistic choice. Consequently the two operators have naturally different semantics and intuitions.

The Peleg sequential composition is then computed as follows

$$\begin{aligned}
(a, \Phi) \in R \circ S \Leftrightarrow \\
\exists_{\varphi'} (a, \varphi') \in R \wedge \exists_{F:W \rightarrow \mathbf{L}^W} \forall_{u \in W} (u, F(u)) \in S \wedge \Phi(w') = \sum_{u \in W} \varphi'(u); F(u)(w')
\end{aligned}$$

and thus $(a, \phi), (a, \rho) \in R \circ S$ with weighted sets ϕ and ρ defined, respectively, as

$$\begin{aligned}
\phi(a) &= 1_{\circ}(a); 1_{\circ}(a) = 1 \\
\rho(b) &= 1_{\circ}(a); \varphi_3(b) = 1; \delta = \delta
\end{aligned}$$

Moreover,

$$\begin{aligned}
(b, \Sigma) \in R \circ S \Leftrightarrow \\
\exists_{\varphi'} (a, \varphi') \in R \wedge \exists_{F:W \rightarrow \mathbf{L}^W} \forall_{u \in W} (u, F(u)) \in S \wedge \Sigma(w') = \sum_{u \in W} \varphi'(u); F(u)(w')
\end{aligned}$$

and thus $(b, \sigma), (b, \theta) \in R \circ S$ with weighted sets σ and θ respectively defined as

$$\begin{aligned}
\sigma(a) &= \varphi_2(a); 1_{\circ}(a) = \gamma; 1 = \gamma \\
\theta(b) &= \varphi_2(a); \varphi_3(b) = \gamma; \delta
\end{aligned}$$

We obtain $R \circ S = \{(a, \phi), (a, \rho), (b, \sigma), (b, \theta)\} \neq R \circ S$.

We use the same notation of Peleg composition to make a clearer comparison between weighted sets. Note that ϕ' and ρ' are additional weighted sets that are in $R \circ S$ and are not in $R \circ S$. This is because a pair only belongs to Peleg composition if there exists a 'path' through both elements a and b , formally

$\forall u \in W \cdot (u, S(u)(w')) \in S$, while Kleisli composition is weaker. In the former, it is forced that both a and b have to be related with some weighted set in S , which is not the case (only a is related with some set), and thus the path through φ_1 does not occur in the computation. In the latter, every ‘path’ should be considered and that is why $\varphi_1(a)$ is taken as an additional path in the computation, resulting in the additional weighted sets ϕ' and ρ' . The pair (a, ω) appears also as an additional path in the Kleisli composition, since the computation allows a ‘path’ through φ_1 to construct a weighted set ω such that $\omega(a), \omega(b) > 0$. An analogous reasoning can be done for w' and τ .

Considering that we follow similar motivations as Peleg on the use of binary multirelations as a semantics for dynamic logics, the framework that we introduce in **Part 2** uses our generalisation of Peleg sequential composition (102). From now on, we refer to this composition simply as the sequential composition of weighted multirelations.

The set $M^{\mathbf{L}}(W)$ of weighted multirelations over \mathbf{L} , together with a set of operators (including \circ and \uplus), provide a suitable semantic structure for \mathcal{F}_2 computations.

Definition 6.2.4 (Algebra of weighted multirelations). *Given a set W and a complete right residuated lattice \mathbf{L} , the algebra of weighted multirelations over \mathbf{L} is the structure*

$$\mathbb{M} = (M^{\mathbf{L}}(W), \cup, \circ, \uplus, \emptyset, 1_{\circ}, 1_{\uplus})$$

where:

- \cup is the binary set union;
- \circ and \uplus correspond to sequential (102) and parallel (100) compositions of weighted multirelations, respectively;
- $1_{\circ} = \{(w, \delta_w) \mid w \in W\}$ is the identity of \circ , with $\delta_w : W \rightarrow \mathbf{L}$ defined as

$$\delta_w(w') = \begin{cases} \top & \text{if } w' = w \\ \perp & \text{otherwise} \end{cases}$$

- $1_{\uplus} = \{(w, \underline{0}) \mid w \in W\}$ is the identity of \uplus , where $\underline{0} : W \rightarrow \mathbf{L}$ is the weighted set defined as $\underline{0}(w) = \perp, \forall w \in W$;

In general, notation \underline{l} stands for the constant function that returns $l \in \mathbf{L}$ for every input.

In the context of program semantics, multirelation 1_{\circ} can be regarded as the semantics of program statement **skip**, a common constructor in imperative programming languages.

In **Part 1** we proved that the algebra of weighted relations is a Kleene algebra. On the other hand, for the case of weighted multirelations, we mentioned before that some axioms of Kleene algebra do not hold, and thus, the algebra of Definition 6.2.4 does not define such a structure. Our next result establishes that it defines, instead, a weaker version of a semiring called *proto-trioid*, i.e. a structure where $(M^{\mathbf{L}}(W), \cup, \emptyset)$

and $(M^{\mathbf{L}}(W), \uplus, 1_{\uplus})$ are two monoids, and $(M^{\mathbf{L}}(W), \circ, 1_{\circ})$ is a structure such that \circ is not required to be associative neither distribute over \cup .

Theorem 6.2.1. *For any complete right residuated lattice \mathbf{L} , \mathbb{M} is a proto-trioid, i.e. an algebra satisfying the following axioms:*

$$R \cup (S \cup T) = (R \cup S) \cup T \quad (103)$$

$$R \cup S = S \cup R \quad (104)$$

$$R \cup R = R \quad (105)$$

$$R \cup \emptyset = R \quad (106)$$

$$(R \circ S) \circ T \subseteq R \circ (S \circ T) \quad (107)$$

$$R \circ 1_{\circ} = 1_{\circ} \circ R = R \quad (108)$$

$$(R \circ S) \cup (R \circ T) \subseteq R \circ (S \cup T) \quad (109)$$

$$(R \cup S) \circ T = (R \circ T) \cup (S \circ T) \quad (110)$$

$$R \circ \emptyset = \emptyset \circ R = \emptyset \quad (111)$$

$$R \uplus (S \uplus T) = (R \uplus S) \uplus T \quad (112)$$

$$R \uplus S = S \uplus R \quad (113)$$

$$R \uplus R = R \quad (114)$$

$$1_{\uplus} \uplus R = R \quad (115)$$

$$R \uplus (S \cup T) = (R \uplus S) \cup (R \uplus T) \quad (116)$$

Proof.

Let $(M^{\mathbf{L}}(W), \cup, \emptyset)$ be a semilattice with least element \emptyset , thus satisfying axioms (103)-(106). The proofs of axioms (107)-(111) are not straightforward and require particular attention.

(107):

Let $(a, \varphi) \in (R \circ S) \circ T$. Then,

$$\exists_{\varphi'} \cdot (a, \varphi') \in (R \circ S) \wedge \exists_{F:W \rightarrow \mathbf{L}^W} \cdot \forall_{b \in W} \cdot (b, F(b)) \in T \wedge \varphi(u) = \sum_{b \in W} \varphi'(b); F(b)(u)$$

and, similarly,

$$\exists_{\varphi''} \cdot (a, \varphi'') \in R \wedge \exists_{F':W \rightarrow \mathbf{L}^W} \cdot \forall_{a' \in W} \cdot (a', F'(a')) \in S \wedge \varphi'(b) = \sum_{a' \in W} \varphi''(a'); F'(a')(b)$$

Thus,

$$\begin{aligned}
\varphi(u) &= \sum_{b \in W} \left(\sum_{a' \in W} (\varphi''(a'); F'(a')(b)); F(b)(u) \right) \\
&= \sum_{b \in W} \left(\sum_{a' \in W} (\varphi''(a'); F'(a')(b); F(b)(u)) \right) \\
&= \sum_{a' \in W} \left(\sum_{b \in W} (\varphi''(a'); F'(a')(b); F(b)(u)) \right) \\
&= \sum_{a' \in W} (\varphi''(a'); \left(\sum_{b \in W} (F'(a')(b); F(b)(u)) \right)) \\
&= \sum_{a' \in W} (\varphi''(a'); F''(a')(u))
\end{aligned}$$

where $F'' : W \rightarrow \mathbf{L}^W$ is defined by

$$F''(a')(u) = \sum_{b \in W} (F'(a')(b); F(b)(u))$$

Clearly $(a', F''(a')) \in S \circ T$ entails $(a, \varphi) \in R \circ (S \circ T)$.

(108):

Case 1: $\underline{1_o \circ R} \subseteq R$.

Suppose $(a, \varphi) \in 1_o \circ R$. Then,

$$(\exists_{\varphi'} (a, \varphi') \in 1_o) \wedge \exists_{F:W \rightarrow \mathbf{L}^W} \forall_{b \in W} (b, F(b)) \in R \wedge \varphi(u) = \sum_{b \in W} \varphi'(b); F(b)(u)$$

\Leftrightarrow

$$(\exists_{\varphi'} (a, \varphi') \in 1_o) \wedge \exists_{F:W \rightarrow \mathbf{L}^W} \forall_{b \in W} (b, F(b)) \in R \wedge \varphi(u) = F(a)(u)$$

because

$$\begin{aligned}
\varphi(u) &= \sum_{b \in W} \varphi'(b); F(b)(u) \\
&= \varphi'(a); F(a)(u) \\
&= \top; F(a)(u) \\
&= F(a)(u)
\end{aligned}$$

with $(a, F(a)) \in R$. Thus, $(a, \varphi) \in R$.

Case 2: $\underline{R} \subseteq 1_o \circ R$:

Conversely, let $(a, \varphi) \in R$, and define $F : W \rightarrow \mathbf{L}^W$ as $F(b) = \begin{cases} \varphi & \text{if } b = a \\ \underline{0} & \text{otherwise} \end{cases}$

such that $(b, F(b)) \in R$. Thus, φ is a function defined as follows

$$\begin{aligned}\varphi(u) &= \sum_{b \in W} (\varphi'(b); F(b)(u)) \\ &= \sum_{b \in W} (\delta_a(b); F(b)(u)), \\ &\text{with } (a, \sum_{b \in W} (\delta_a(b); F(b))) \in (1_\circ \circ R)\end{aligned}$$

Thus, $(a, \varphi) \in 1_\circ \circ R$. Let us now prove $R \circ 1_\circ = R$.

Case 1: $R \circ 1_\circ \subseteq R$:

Suppose $(a, \varphi) \in R \circ 1_\circ$. Then,

$$\exists_{\varphi'} \cdot (a, \varphi') \in R \wedge \exists_{F:W \rightarrow \mathbf{L}^W} \cdot \forall_{b \in W} \cdot (b, F(b)) \in 1_\circ \wedge \varphi(u) = \sum_{b \in W} (\varphi'(b); \delta_b(u))$$

\Leftrightarrow

$$\exists_{\varphi'} \cdot (a, \varphi') \in R \wedge \exists_{F:W \rightarrow \mathbf{L}^W} \cdot \forall_{b \in W} \cdot (b, F(b)) \in 1_\circ \wedge \varphi(u) = \varphi'(b)$$

because

$$\begin{aligned}\varphi(u) &= \sum_{b \in W} (\varphi'(b); \delta_b(u)) \\ &= \varphi'(b); \delta_b(b) \\ &= \varphi'(b); \top \\ &= \varphi'(b)\end{aligned}$$

with $(a, \varphi') \in R$. Thus $(a, \varphi) \in R$.

Case 2: $R \subseteq R \circ 1_\circ$:

Let $(a, \varphi) \in R$, and define $F : W \rightarrow \mathbf{L}^W$ as $F(b)(u) = \delta_b(u)$. Following an argument similar to the one above,

$$\varphi(u) = \sum_{b \in W} (\varphi(b); F(b)(u)) = \sum_{b \in W} \varphi(b); \delta_b(u)$$

with $(a, \sum_{b \in W} \varphi(b); \delta_b) \in (R \circ 1_\circ)$.

(109):

Assume $(a, \varphi) \in (R \circ S) \cup (R \circ T)$. Then, $(a, \varphi) \in R \circ S \vee (a, \varphi) \in R \circ T$. Thus,

$$\begin{aligned}
& \exists_{\varphi'} \cdot (a, \varphi') \in R \quad \wedge \quad \left(\exists_{F:W \rightarrow \mathbf{L}^W} \cdot \forall_{b \in W} \cdot ((b, F(b)) \in S \vee (b, F(b)) \in T) \right) \\
& \quad \wedge \quad \varphi(u) = \sum_{b \in W} (\varphi'(b); F(b)(u)) \\
& \Leftrightarrow \\
& \exists_{\varphi'} \cdot (a, \varphi') \in R \quad \wedge \quad \left(\exists_{F:W \rightarrow \mathbf{L}^W} \cdot \forall_{b \in W} \cdot (b, F(b)) \in S \cup T \right) \\
& \quad \wedge \quad \varphi(u) = \sum_{b \in W} (\varphi'(b); F(b)(u)) \\
& \Leftrightarrow \\
& (a, \varphi) \in R \circ (S \cup T)
\end{aligned}$$

(110):

We start by proving $(R \cup S) \circ T \subseteq (R \circ T) \cup (S \circ T)$. Let us assume $(a, \varphi) \in (R \cup S) \circ T$. Then

$$\begin{aligned}
& \exists_{\varphi'} \cdot (a, \varphi') \in (R \cup S) \quad \wedge \quad \exists_{F:W \rightarrow \mathbf{L}^W} \cdot \forall_{b \in W} \cdot (b, F(b)) \in T \\
& \quad \wedge \quad \varphi(u) = \sum_{b \in W} (\varphi'(b); F(b)(u)) \\
& \Leftrightarrow \\
& \exists_{\varphi'} \cdot ((a, \varphi') \in R \vee (a, \varphi') \in S) \quad \wedge \quad \exists_{F:W \rightarrow \mathbf{L}^W} \cdot \forall_{b \in W} \cdot (b, F(b)) \in T \\
& \quad \wedge \quad \varphi(u) = \sum_{b \in W} (\varphi'(b); F(b)(u)) \\
& \Leftrightarrow \\
& \left(\exists_{\varphi'} \cdot (a, \varphi') \in R \quad \wedge \quad \exists_{F:W \rightarrow \mathbf{L}^W} \cdot \forall_{b \in W} \cdot (b, F(b)) \in T \quad \wedge \quad \varphi(u) = \sum_{b \in W} (\varphi'(b); F(b)(u)) \right) \\
& \quad \vee \quad \left(\exists_{\varphi'} \cdot (a, \varphi') \in S \quad \wedge \quad \exists_{F:W \rightarrow \mathbf{L}^W} \cdot \forall_{b \in W} \cdot (b, F(b)) \in T \quad \wedge \quad \varphi(u) = \sum_{b \in W} (\varphi'(b); F(b)(u)) \right) \\
& \Leftrightarrow \\
& ((a, \varphi) \in R \circ T) \vee ((a, \varphi) \in S \circ T) \\
& \Leftrightarrow \\
& (a, \varphi) \in (R \circ T) \cup (S \circ T)
\end{aligned}$$

Conversely, assume $(a, \varphi) \in (R \circ T) \cup (S \circ T)$. Then

$$\begin{aligned}
& (a, \varphi) \in R \circ T \vee (a, \varphi) \in S \circ T \\
& \Leftrightarrow \\
& \left(\exists_{\varphi'} \cdot (a, \varphi') \in R \quad \wedge \quad \exists_{F:W \rightarrow \mathbf{L}^W} \cdot \forall_{b \in W} \cdot (b, F(b)) \in T \quad \wedge \quad \varphi(u) = \sum_{b \in W} (\varphi'(b); F(b)(u)) \right)
\end{aligned}$$

$$\begin{aligned}
& \vee (\exists_{\varphi'} \cdot (a, \varphi') \in S \wedge \exists_{F:W \rightarrow \mathbf{L}^W} \cdot \forall_{b \in W} \cdot (b, F(b)) \in T \wedge \varphi(u) = \sum_{b \in W} (\varphi'(b); F(b)(u))) \\
& \Leftrightarrow \\
& \exists_{\varphi'} \cdot ((a, \varphi') \in R \vee (a, \varphi') \in S) \wedge \exists_{F:W \rightarrow \mathbf{L}^W} \cdot \forall_{b \in W} \cdot (b, F(b)) \in R \wedge \sum_{b \in W} (\varphi'(b); F(b)(u)) \\
& \Leftrightarrow \\
& \exists_{\varphi'} \cdot (a, \varphi') \in R \cup S \wedge \exists_{F:W \rightarrow \mathbf{L}^W} \cdot \forall_{b \in W} \cdot (b, F(b)) \in T \wedge \sum_{b \in W} (\varphi'(b); F(b)(u)) \\
& \Leftrightarrow \\
& (a, \varphi) \in (R \cup S) \circ T
\end{aligned}$$

(111):

Let us prove $\emptyset \circ R \subseteq \emptyset$. Suppose $(a, \varphi) \in \emptyset \circ R$. Then,

$$\exists_{\varphi'} \cdot (a, \varphi') \in \emptyset \wedge \exists_{F:W \rightarrow \mathbf{L}^W} \cdot (b, F(b)) \in R \wedge \varphi(u) = \sum_{b \in W} (\varphi'(b); F(b)(u))$$

But $\nexists_{\varphi'} \cdot (a, \varphi') \in \emptyset$, since \emptyset is the empty set. So there is no $(a, \varphi) \in \emptyset \circ R$, and thus $(a, \varphi) \in \emptyset$.

To prove that $R \circ \emptyset \subseteq \emptyset$, assume $(a, \varphi) \in R \circ \emptyset$. Then,

$$\exists_{\varphi'} \cdot (a, \varphi') \in R \wedge \exists_{F:W \rightarrow \mathbf{L}^W} \cdot \forall_{b \in W} \cdot (b, F(b)) \in \emptyset \wedge \varphi(u) = \sum_{b \in W} (\varphi'(b); F(b))$$

But there is no $(b, F(b)) \in \emptyset$ since \emptyset is the empty set. Hence $(a, \varphi) \in \emptyset$. The two converse inclusions are trivial.

(112)-(114):

These axioms come directly from (100) and the properties of \cup on right residuated lattices.

(115):

Consider an arbitrary weighted multirelation R over a set of states W . Then,

$$\begin{aligned}
& R \uplus 1_{\uplus} \\
& = \quad \{ \text{definition of } \uplus \text{ (100) and } 1_{\uplus} \} \\
& \quad \{(a, \varphi_R \cup \{(a, \emptyset) \mid a \in W\}) \mid (a, \varphi_R) \in R\} \\
& = \quad \{ \text{for any } a \in W, \varphi_R(a) + 0 \text{ by (40)} \} \\
& \quad \{(a, \varphi_R) \mid (a, \varphi_R) \in R\} \\
& = \quad \{ \text{identity} \} \\
& R
\end{aligned}$$

(116):

First we prove $R \uplus (S \cup T) \subseteq R \uplus S \cup R \uplus T$. Assume $(a, \varphi) \in R \uplus (S \cup T)$. Then,

$$\begin{aligned}
& \exists_{\varphi_1, \varphi_2} \cdot (a, \varphi_1) \in R \wedge (a, \varphi_2) \in S \cup T \wedge \varphi = \varphi_1 + \varphi_2 \\
\Rightarrow & \\
& \exists_{\varphi_1, \varphi_2} \cdot \left(((a, \varphi_1) \in R \wedge (a, \varphi_2) \in S) \vee ((a, \varphi_1) \in R \wedge (a, \varphi_2) \in T) \right) \wedge \varphi = \varphi_1 + \varphi_2 \\
\Rightarrow & \\
& ((a, \varphi) \in R \uplus S) \vee ((a, \varphi) \in R \uplus T) \\
\Rightarrow & \\
& (a, \varphi) \in (R \uplus S) \cup (R \uplus T)
\end{aligned}$$

Conversely, suppose that $(a, \varphi) \in (R \uplus S) \cup (R \uplus T)$. Then,

$$\begin{aligned}
& (a, \varphi) \in R \uplus S \vee (a, \varphi) \in R \uplus T \\
\Rightarrow & \\
& \exists_{\varphi_1, \varphi_2} \cdot \left(((a, \varphi_1) \in R \wedge (a, \varphi_2) \in S) \vee ((a, \varphi_1) \in R \wedge (a, \varphi_2) \in T) \right) \wedge \varphi = \varphi_1 + \varphi_2 \\
\Rightarrow & \\
& \exists_{\varphi_1, \varphi_2} \cdot \left((a, \varphi_1) \in R \wedge ((a, \varphi_2) \in S \vee (a, \varphi_2) \in T) \right) \wedge \varphi = \varphi_1 + \varphi_2 \\
\Rightarrow & \\
& \exists_{\varphi_1, \varphi_2} \cdot \left((a, \varphi_1) \in R \wedge (a, \varphi_2) \in S \cup T \right) \wedge \varphi = \varphi_1 + \varphi_2 \\
\Rightarrow & \\
& (a, \varphi) \in R \uplus (S \cup T)
\end{aligned}$$

□

Note that due to the definition of sequential composition (102), axioms (107) and (109) are weaker than the usual equalities of associativity and left distributivity which characterise a semiring embedded in a Kleene algebra. The next counter examples show precisely $R \circ (S \circ T) \not\subseteq (R \circ S) \circ T$ and $R \circ (S \cup T) \not\subseteq (R \circ S) \cup (R \circ T)$.

First, to prove $R \circ (S \circ T) \not\subseteq (R \circ S) \circ T$, consider the weighted multirelation $R = \{(a, \varphi_1), (a, 1_\circ), (b, \varphi_2)\}$, where $\varphi_1(a) = \alpha$, $\varphi_1(b) = \beta$, $\varphi_2(a) = \gamma$ and 0 otherwise, and the weighted multirelation $S = \{(a, 1_\circ), (a, \varphi_3)\}$, where $\varphi_3(b) = \delta$ and 0 otherwise, with $\alpha, \beta, \gamma, \delta \in \mathbf{L}$. We show $R \circ (R \circ S) \not\subseteq (R \circ R) \circ S$. First let us compute $R \circ R$.

We know, by (102),

$$\begin{aligned}
& (a, \Phi) \in R \circ R \Leftrightarrow \\
& \exists_{\varphi'} \cdot (a, \varphi') \in R \wedge \exists_F \cdot \forall_{u \in W} \cdot (u, F(u)) \in R \wedge \Phi(w') = \sum_{u \in W} \varphi'(u); F(u)(w')
\end{aligned}$$

and thus $(a, \phi), (a, \rho) \in R \circ R$ with weighted set ϕ defined such that

$$\begin{aligned}\phi(a) &= \varphi_1(a); 1_{\circ}(a) + \varphi_1(b); \varphi_2(a) + 1_{\circ}(a); 1_{\circ}(a) \\ &= \alpha; 1 + \beta; \gamma + 1; 1 \\ &= \alpha + \beta; \gamma\end{aligned}$$

and ρ such that

$$\begin{aligned}\rho(a) &= \varphi_1(a); \varphi_1(a) + \varphi_1(b); \varphi_2(a) + 1_{\circ}(a); \varphi_1(a) \\ &= \alpha; \alpha + \beta; \gamma + 1; \alpha \\ &= \alpha; \alpha + \beta; \gamma + \alpha\end{aligned}$$

$$\begin{aligned}\rho(b) &= \varphi_1(a); \varphi_1(b) + 1_{\circ}(a); \varphi_1(b) \\ &= \alpha; \beta + 1; \beta \\ &= \alpha; \beta + \beta\end{aligned}$$

Analogously,

$$\begin{aligned}(b, \Sigma) \in R \circ R &\Leftrightarrow \\ \exists \varphi' \cdot (b, \varphi') \in R \wedge \exists_F \cdot \forall_{u \in W} \cdot (u, F(u)) \in R \wedge \Sigma(w') &= \sum_{u \in W} \varphi'(u); F(u)(w')\end{aligned}$$

and thus $(b, \sigma), (b, \theta) \in R \circ R$ with σ and θ respectively defined as

$$\sigma(a) = \varphi_2(a); 1_{\circ}(a) = \gamma; 1 = \gamma$$

and

$$\begin{aligned}\theta(a) &= \varphi_2(a); \varphi_1(a) = \gamma; \alpha \\ \theta(b) &= \varphi_2(a); \varphi_1(b) = \gamma; \beta\end{aligned}$$

Hence $R \circ R = \{(a, \phi), (a, \rho), (b, \sigma), (b, \theta)\}$.

To compute $(R \circ R) \circ S$, we know

$$(a, \Phi) \in (R \circ R) \circ S \Leftrightarrow$$

$$\exists_{\Phi} \cdot (a, \Phi) \in R \circ R \wedge \exists_F \cdot \forall_{u \in W} \cdot (u, F(u)) \in S \wedge \Phi(w') = \sum_{u \in W} \Phi(u); F(u)(w')$$

and therefore $(a, \varphi), (a, \varrho) \in (R \circ R) \circ S$ with φ, ϱ respectively defined as

$$\varphi(a) = \phi(a); 1_{\circ}(a) = (\alpha + \beta; \gamma); 1 = \alpha + \beta; \gamma$$

and

$$\varrho(b) = \phi(a); \varphi_3(b) = (\alpha + \beta; \gamma); \delta$$

And analogously,

$$(b, \Sigma) \in (R \circ R) \circ S \Leftrightarrow$$

$$\exists_{\Phi} \cdot (b, \Phi) \in R \circ R \wedge \exists_F \cdot \forall_{u \in W} \cdot (u, F(u)) \in S \wedge \Sigma(w') = \sum_{u \in W} \Phi(u); F(u)(w')$$

and thus $(b, \zeta), (b, \vartheta) \in (R \circ R) \circ S$ with weighted sets ζ, ϑ respectively defined as

$$\zeta(a) = \sigma(a); 1_{\circ}(a) = \gamma; 1 = \gamma$$

and

$$\vartheta(b) = \sigma(a); \varphi_3(b) = \gamma; \delta$$

Hence $(R \circ R) \circ S = \{(a, \varphi), (a, \varrho), (b, \zeta), (b, \vartheta)\}$.

To compute $R \circ (R \circ S)$, we already obtained, from the example above,

$R \circ S = \{(a, \phi), (a, \rho), (b, \sigma), (b, \theta)\}$, with $\phi(a) = 1, \rho(b) = \delta, \sigma(a) = \gamma, \theta(b) = \gamma; \delta$.

Now, to compute $R \circ (R \circ S)$, we know

$$(a, \Phi) \in R \circ (R \circ S) \Leftrightarrow$$

$$\exists_{\varphi'} \cdot (a, \varphi') \in R \wedge \exists_F \cdot \forall_{u \in W} \cdot (u, F(u)) \in R \circ S \wedge \Phi(w') = \sum_{u \in W} \varphi'(u); F(u)(w')$$

and therefore $(a, \varphi), (a, \varrho'), (a, \varepsilon) \in R \circ (R \circ S)$ with weighted sets φ, ϱ' and ε defined as

$$\varphi(a) = 1_{\circ}(a); \phi(a) + \varphi_1(a); \phi(a) + \varphi_1(b); \sigma(a) = 1; 1 + \alpha; 1 + \beta; \gamma = \alpha + \beta; \gamma$$

$$\begin{aligned}
\varrho'(b) &= 1_{\circ}(a); \rho(b) + \varphi_1(a); \rho(b) + \varphi_1(b); \theta(b) \\
&= 1; \delta + \alpha; \delta + \beta; (\gamma; \delta) \\
&= \delta + \alpha; \delta + \beta; (\gamma; \delta)
\end{aligned}$$

and

$$\begin{aligned}
\varepsilon(a) &= \varphi_1(a); \phi(a) + \varphi_1(b); \sigma(a) = \alpha; 1 + \beta; \gamma = \alpha + \beta; \gamma \\
\varepsilon(b) &= \varphi_1(a); \rho(b) + \varphi_1(b); \theta(b) = \alpha; \delta + \beta; (\gamma; \delta)
\end{aligned}$$

Additionally,

$$\begin{aligned}
(b, \Sigma) \in R \circ (R \circ S) &\Leftrightarrow \\
\exists_{\varphi'} \cdot (a, \varphi') \in R \wedge \exists_F \cdot \forall_{u \in W} \cdot (u, F(u)) \in R \circ S \wedge \Sigma(w') &= \sum_{u \in W} \varphi'(u); F(u)(w')
\end{aligned}$$

and therefore $(b, \zeta), (b, \vartheta) \in R \circ (R \circ S)$ with weighted sets ζ and ϑ defined as

$$\begin{aligned}
\zeta(a) &= \varphi_2(a); \phi(a) = \gamma; 1 = \gamma \\
\vartheta(b) &= \varphi_2(a); \rho(b) = \gamma; \delta
\end{aligned}$$

Hence,

$$R \circ (R \circ S) = \{(a, \varphi), (a, \varrho'), (a, \varepsilon), (b, \zeta), (b, \vartheta)\} \not\subseteq \{(a, \varphi), (a, \varrho), (b, \zeta), (b, \vartheta)\} = (R \circ R) \circ S$$

Now let us discuss axiom (109). Consider multirelation $R = \{(a, \varphi_1)\}$ where $\varphi_1(a) = \alpha$, $\varphi_1(b) = \beta$ and 0 otherwise, multirelation $S = \{(a, 1_{\circ})\}$ and $T = \{(b, 1_{\circ})\}$. We prove $R \circ (S \cup T) \not\subseteq (R \circ S) \cup (R \circ T)$. By (102),

$$\begin{aligned}
(a, \Phi) \in R \circ (S \cup T) &\Leftrightarrow \\
\exists_{\varphi'} \cdot (a, \varphi') \in R \wedge \exists_F \cdot \forall_{u \in W} \cdot (u, F(u)) \in S \cup T \wedge \Phi(w') &= \sum_{u \in W} \varphi'(u); F(u)(w')
\end{aligned}$$

and thus $S \cup T = \{(a, 1_{\circ}), (b, 1_{\circ})\}$, and $R \circ (S \cup T)$ is obtained, using (102), as a multirelation with weighted set ϕ defined as follows

$$\phi(a) = \varphi_1(a); 1_\circ(a) = \alpha; 1 = \alpha$$

$$\phi(b) = \varphi_1(b); 1_\circ(b) = \beta; 1 = \beta$$

i.e. $R \circ (S \cup T) = \{(a, \phi)\} = \{(a, \varphi_1)\} = R$.

On the other hand, $R \circ S = R \circ T = \{\}$ and therefore $(R \circ S) \cup (R \circ T) = \{\}$, from which we conclude $R \circ (S \cup T) \not\subseteq (R \circ S) \cup (R \circ T)$.

A possible and obvious continuation to this study is to define the Parikh sequential composition [FKST17], and compare it with the ones of Kleisli and Peleg. Even if approached, at least in a first attempt, as a mere theoretical exercise, such a study would have potential to enrich the research path that we pursue in **Part 2** of this thesis, by giving intuitions to the different definitions in the program semantics that we present in the next chapter. Even if a discussion on the differences between Kleisli and Peleg was started, a more extensive study on their properties, as well as the inclusion of Parikh composition, is lacking. In [FKST17], the authors state that Kleisli composition is associative, contrary to the one of Peleg, but does not have unit. On the other hand, Parikh composition is not associative, although it is associative for a specific class of multirelations. It is this kind of study that we intend to make in the future for weighted multirelations.

A WEIGHTED *MULTI-FLOW* SEMANTICS

The second component, the formal semantics of weighted “multi-flow” computations, is given, as in Chapter 4, by interpreting of expressions in the language \mathcal{F}_2 , generated by grammar of Table 4, over states. We parametrize such a semantics by a complete right residuated lattice, which plays a double role, to interpret programs and to give meaning to variables. Let us fix a complete right residuated lattice \mathbf{L} , and a set of variables X . *Program states* are graded valuations of variables, i.e. functions

$$w : X \rightarrow \mathbf{L}^{\mathbb{R}}$$

We denote the set of all states by $\mathbf{L}^{X^{\mathbb{R}}}$.

Analogously to Chapter 4, to simplify reasoning about \mathcal{F}_2 -programs at a later stage, it is useful to decompose conditional and switch statements into more elementary operators, namely tests, as well as parallel and sequential composition of programs. Such operators, although not explicitly part of the syntax, can easily be added to the language through an additional rule:

$$\pi ::= \mathbf{skip} \mid \pi_0 \mid \rho? \mid \pi; \pi \mid \pi \parallel \pi \quad (117)$$

where $\pi_0 \in \text{Prog}_0$, and $\rho?$ stands for a notion of a *test*. The latter depends, of course, on the space of truth values considered in the semantics, i.e. on the particular choice of the right residuated lattice \mathbf{L} . Hence, a conditional statement

if p then π_1 else π_2 endif

is encoded as

$$p?; \pi_1 \parallel (p \rightarrow \perp)?; \pi_2$$

where $p \rightarrow \perp$ denotes the “negation” of predicate p . Similarly to what was done in **Part 1**, in order to support a weighted truth space to evaluate a predicate, negation is not explicitly denoted, but defined instead in terms of the operator \rightarrow and the least element of \mathbf{L} . The formal definition of tests, together with the semantics of operator \rightarrow , will be given in the next chapter when defining satisfaction relation for a suitable dynamic logic. Note that in the encoding of the conditional above the parallel composition plays the role usually assigned to the choice operator $+$ in the decomposition of conditionals in classical

imperative languages. The motivation for this definition is to suitably capture the behaviour of conditionals in \mathcal{F}_2 -programs, as intuitively explained in the Introduction.

To illustrate the proposed semantics we will resort, along the next chapters, to an excerpt of a FAS program, taken from [VMA10].

Example 7.0.1.

```
O2_low:=FUZZY SET((70,0),(75,1),(85,1),(90,0))
```

```
if O2 is in O2_low then PIP_inc:=5 else PIP_inc:=0
```

The program represents an excerpt of a module of a medical fuzzy control system which supports mechanically ventilated patients after cardiac surgery. Operationally, it suggests a value for the *peak inspiratory pressure* (PIP) based on the O_2 level measured in a patient after a cardiac surgery. The code instantiates, in the first line, the variable `O2_low` to a weighted set, as will be explained below. The next instruction defines a fuzzy control rule, encoded in a `if-then-else` statement. Depending on the oxygen level of a patient, the rule makes a modification to the peak inspiratory pressure (`PIP_inc`).

In this setting, functional and predicate terms are interpreted as follows.

Definition 7.0.1 (Interpretation of functional terms). *Let F be a set of function symbols and X a set of variables. The interpretation of terms $t \in T_F(X)$ in a state $w \in W \subseteq \mathbf{L}^{X^{\mathbb{R}}}$ is given by the map*

$$\llbracket _ \rrbracket_w : T_F(X) \rightarrow \mathbf{L}^{\mathbb{R}}$$

recursively defined as follows:

- $\llbracket x \rrbracket_w(r) = w(x)(r)$
- $\llbracket c \rrbracket_w(r) = \delta_c(r) = \begin{cases} \top & \text{if } r = c \\ \perp & \text{otherwise} \end{cases}$
- $\llbracket (t_1, \dots, t_n) \rrbracket_w(r) = \sum_{i \in I} \{ \prod_{j=1}^n \llbracket t_j \rrbracket_w(r_j^i) \mid f(r_1^i, \dots, r_n^i) = r \}$

where I is the cardinality of the set of all possible solutions of $f(r_1^i, \dots, r_n^i) = r$ in \mathbb{R} , with each f of arity n being interpreted as a function on real numbers $\mathbb{R}^n \rightarrow \mathbb{R}$ (e.g. $+$, \times , 2 , $\sqrt{}$, \dots), $x \in X$.

Let us illustrate this semantics resorting to Example 7.0.1, and choosing \mathbf{L} as the complete right residuated lattice \mathbf{G} of Example 2.2.7.

The statement

```
O2_low:=FUZZY SET((70,0),(75,1),(85,1),(90,0))
```

assigns a fuzzy set to the variable `O2_low`. The notation used means that `O2_low` is defined, in a given state, as a weighted set, linear on the open intervals $]70, 75[$, $]75, 85[$, $]85, 90[$ and constant on $] - \infty, 70[$ and $]90, +\infty[$, and for each $r \in \mathbb{R}$, $O2_low(r)$ coincides either with the left limit or the right limit of `O2_low` at r . Formally,

$$[[O2_low]]_{w_0}(r) = w_0(O2_low)(r) = \begin{cases} r/5 - 14 & \text{if } 70 \leq r \leq 75 \\ 1 & \text{if } 75 \leq r \leq 85 \\ -r/5 + 18 & \text{if } 85 \leq r \leq 90 \\ 0 & \text{otherwise} \end{cases}$$

where w_0 is the attribution state of `O2_low` defined above, and for each $r \in \mathbb{R}$, as plotted in Figure 11.

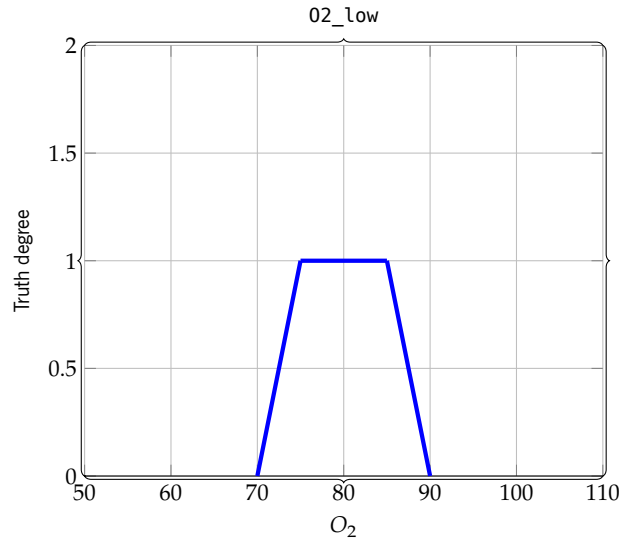


Figure 11: Representation of the variable `O2_low`

Intuitively it gives, for each oxygen value, a degree which measures how “low” such value actually is. In other words, it defines numerically what it means for the variable to represent a low level of oxygen concentration.

Consider now computing the value of term `O2_low+5` in state w_0 . According to Definition 7.0.1, the term 5 is defined as

$$[[5]]_{w_0}(r) = w_0(5)(r) = \begin{cases} 1 & \text{if } r = 5 \\ 0 & \text{otherwise} \end{cases}$$

The semantics of term `O2_low+5` in state w_0 is computed as follows:

$$[[O2_low+5]]_{w_0}(r) = \sum_{i \in I} \{ [[O2_low]]_{w_0}(r_1^i); [[5]]_{w_0}(r_2^i) \mid r_1^1 + r_2^2 = r \}$$

$$= \llbracket 02_low \rrbracket_{w_0}(r_1^1); \llbracket 5 \rrbracket_{w_0}(r_2^1)$$

where, for concrete values of (r_1^1, r_2^1) , the vertices are computed as

$$\begin{aligned} \llbracket 02_low+5 \rrbracket_{w_0}(75) &= \llbracket 02_low \rrbracket_{w_0}(70); \llbracket 5 \rrbracket_{w_0}(5) = \min\{0, 1\} = 0 \\ \llbracket 02_low+5 \rrbracket_{w_0}(80) &= \llbracket 02_low \rrbracket_{w_0}(75); \llbracket 5 \rrbracket_{w_0}(5) = \min\{1, 1\} = 1 \\ \llbracket 02_low+5 \rrbracket_{w_0}(90) &= \llbracket 02_low \rrbracket_{w_0}(85); \llbracket 5 \rrbracket_{w_0}(5) = \min\{1, 1\} = 1 \\ \llbracket 02_low+5 \rrbracket_{w_0}(95) &= \llbracket 02_low \rrbracket_{w_0}(90); \llbracket 5 \rrbracket_{w_0}(5) = \min\{0, 1\} = 0 \end{aligned}$$

For an intermediate value of r , e.g. $r = 77$, we have

$$\llbracket 02_low+5 \rrbracket_{w_0}(77) = \llbracket 02_low \rrbracket_{w_0}(72); \llbracket 5 \rrbracket_{w_0}(5) = \min\{0.4, 1\} = 0.4$$

Note that, in this scenario, for each one of the intermediate r in the interval $]70, 90[$, there is only one solution (r_1^1, r_2^1) (i.e. $i = 1$) to the equation $r = r_1^i + r_2^i$. For example, for $r = 75$, another possible solution ($i = 2$) to the equation $75 = r_1^2 + r_2^2$ could hypothetically be $(r_1^2, r_2^2) = (65, 10)$, but obviously $\llbracket 02_low \rrbracket_{w_0}(65); \llbracket 5 \rrbracket_{w_0}(10) = 0; 0 = 0$. Moreover, for other values of r , e.g. $r = 70$, it is easy to verify that there is no pair (r_1^i, r_2^i) such that $\llbracket 02_low+5 \rrbracket_{w_0}(70) = \llbracket 02_low \rrbracket_{w_0}(r_1^i); \llbracket 5 \rrbracket_{w_0}(r_2^i) > 0$. Therefore $\llbracket 02_low+5 \rrbracket_{w_0}(r) = 0$ for $r \notin]70, 90[$. Note that the interpretation of the term 02_low+5 in w_0 is given by a horizontal translation of the graph of 02_low , as depicted in Figure 12.

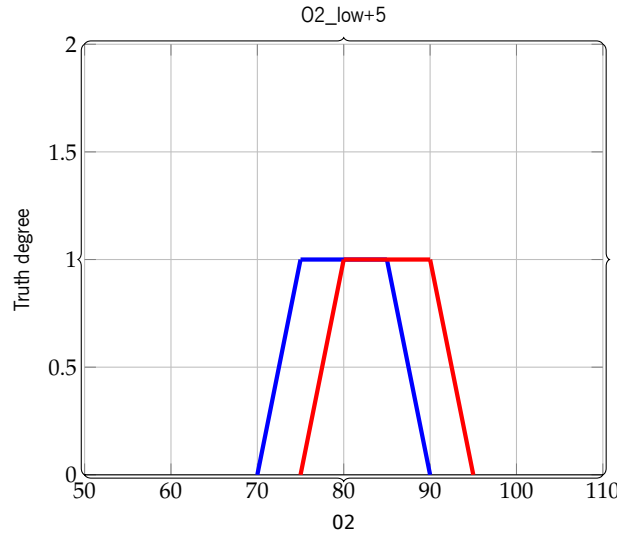


Figure 12: Graph 02_low (blue line) and 02_low+5 (red line)

Definition 7.0.2 (Interpretation of predicate terms). *Let P be a set of predicate symbols and X a set of variables. The interpretation of a predicate $p \in T_P(X)$ in state $w \in W$ is given by the map*

$$\llbracket _ \rrbracket_w : T_P(X) \rightarrow \mathbf{L}$$

defined by

$$\llbracket p(t_1, \dots, t_n) \rrbracket_w = \sum_{i \in I} \left\{ \prod_{j=1}^n \llbracket t_j \rrbracket_w(r_j^i) \mid p(r_1^i, \dots, r_n^i) \right\}$$

where I is the cardinality of the set of all possible values $(r_1^i, \dots, r_n^i) \in \mathbb{R}^n$ satisfying $p(r_1^i, \dots, r_n^i)$, with p of arity n interpreted as a predicate.

As an example, let us compute the interpretation of the predicate $02_low \leq 75$ in the state w_0 described previously, according to Definition 7.0.2.

$$\begin{aligned} \llbracket 02_low \leq 75 \rrbracket_{w_0} &= \sum_{i \in I} \left\{ \prod_{j=1}^2 \llbracket t_j \rrbracket_{w_0}(r_j^i) \mid r_1^i \leq r_2^i \right\} \\ &= \sum_{i \in I} \left\{ \llbracket 02_low \rrbracket_{w_0}(r_1^i); \llbracket 75 \rrbracket_{w_0}(r_2^i) \mid r_1^i \leq r_2^i \right\} \\ &= \llbracket 02_low \rrbracket_{w_0}(75); \llbracket 75 \rrbracket_{w_0}(75) + \dots + \llbracket 02_low \rrbracket_{w_0}(70); \llbracket 75 \rrbracket_{w_0}(75) \\ &= 1; 1 + \dots + 0; 1 \\ &= \max\{\min\{1, 1\}, \dots, \min\{0, 1\}\} = 1 \end{aligned}$$

Note that the values of terms in ... are lower than 1.

Let us consider another example, consisting of the variation of the human body temperature along a 24h period [HWL⁺08, MW81] of two groups of individuals: the first group is composed by individuals with standard sleep schedule, from 0h to 7h (blue line), while the second is composed by subjects which a reduced sleep time, from 0h to 4h (red line). The functions corresponding to each group are, respectively, the polynomials of degree 5

$$\text{temp}(t) = -0.000019484t^5 + 0.00117855t^4 - 0.0256008t^3 + 0.233205t^2 - 0.706115t + 37$$

and

$$\text{temp_red}(t) = -0.0000131385t^5 + 0.000755533t^4 - 0.015809t^3 + 0.143592t^2 - 0.491162t + 37$$

obtained by doing a polynomial interpolation on a data set of values obtained from [HWL⁺08, MW81]. Their plots are represented by the figure below.

We can observe that, in general, the temperature is lower in individuals who were tied to an altered sleep rhythm. Suppose now that we want to evaluate, using the semantics introduced in this chapter, the predicate $\text{temp} \leq \text{temp_red}$, i.e. how normal is the temperature during the interval when the predicate is 'classically' true. Such period corresponds precisely to the sleeping period of the second group of individuals.

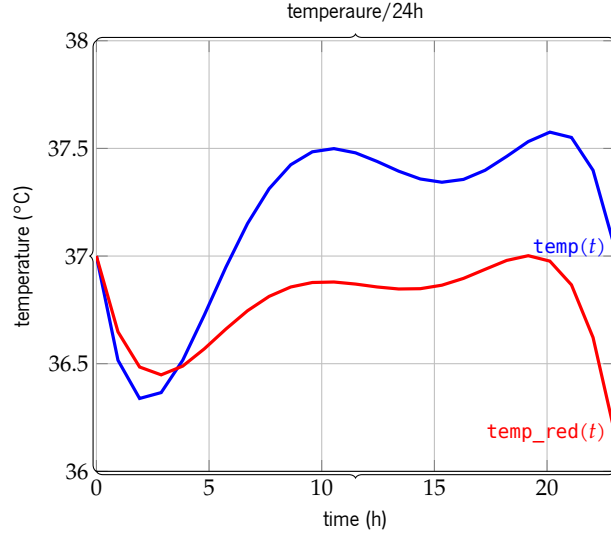


Figure 13: Human body temperature variation in a 24h period

In order to adapt the variables to the introduced semantics, so we can evaluate the predicate in one of the lattices presented before, it is necessary make a correspondence from the interval $[35, 38]$ to the real interval $[0, 1]$. A temperature below 35°C is considered *Hypothermia* and above 37.5°C is considered *Hyperthermia* or *fever*. Hence, in the presented semantics, on the one hand 0 means that the temperature is close to Hypothermia, and a value close to 0 is considered a low temperature; on the other hand, a value close to 1 is considered fever. Naturally, the values in between mean a normal temperature, which varies in function of the hour of the day.

Note that there is a small period that the temperature is slightly above 37.5°C for individuals on the first group, but rapidly decreases. Even though we could consider a temperature above 37.5°C to be fever, the values represented may be slightly different each day, and vary from person to person, and thus such a small variation may not be considered a fever condition.

Consider the state w_0 as the state where variables `temp` and `temp_red` are the polynomials of degree 5 defined above. The evaluation of the predicate `temp ≤ temp_red` in w_0 corresponds, according to Definition 7.0.2, to the following computation

$$\begin{aligned}
\llbracket \text{temp} \leq \text{temp_red} \rrbracket_{w_0} &\stackrel{\text{def}}{=} \sum_{i \in I} \left\{ \prod_{j=1}^2 \llbracket t_j \rrbracket_{w_0}(r_j^i) \mid r_1^i \leq r_2^i \right\} \\
&= \sum_{i \in I} \left\{ \llbracket \text{temp} \rrbracket_{w_0}(r_1^i); \llbracket \text{temp_red} \rrbracket_{w_0}(r_2^i) \mid r_1^i \leq r_2^i \right\} \\
&= \llbracket \text{temp} \rrbracket_{w_0}(3.61685); \llbracket \text{temp_red} \rrbracket_{w_0}(3.61685) + \\
&\quad \dots + \llbracket \text{temp} \rrbracket_{w_0}(0); \llbracket \text{temp_red} \rrbracket_{w_0}(0) \\
&= \max\{\min\{\llbracket \text{temp} \rrbracket_{w_0}(3.61685), \llbracket \text{temp_red} \rrbracket_{w_0}(3.61685)\}, \\
&\quad \dots, \min\{\llbracket \text{temp} \rrbracket_{w_0}(0), \llbracket \text{temp_red} \rrbracket_{w_0}(0)\}\} \\
&= \llbracket \text{temp} \rrbracket_{w_0}(0)
\end{aligned}$$

By solving the equation $\text{temp}(t) = \text{temp_red}(t)$, we obtain $t \approx 3.61685$ as one of the relevant solutions. It is possible to observe that the local maximum in the interval $[0, 3.61685]$ is at $0h$, and therefore the truth degree of the predicate is $\llbracket \text{temp} \rrbracket_{w_0}(0)$, which corresponds to $\text{temp}(0) = 37^\circ\text{C}$.

Let us consider now another example, which the introduced semantics is also able to capture, consisting on alterations in diurnal variation in diastolic blood pressure in hypertensive patients, which consume alcohol regularly. The comparison was made between a control period (without alcohol consumption) and a period with regular alcohol consumption. The data, measured once each two hours, was obtained from [KAK⁺96], and is depicted in Figure 14.

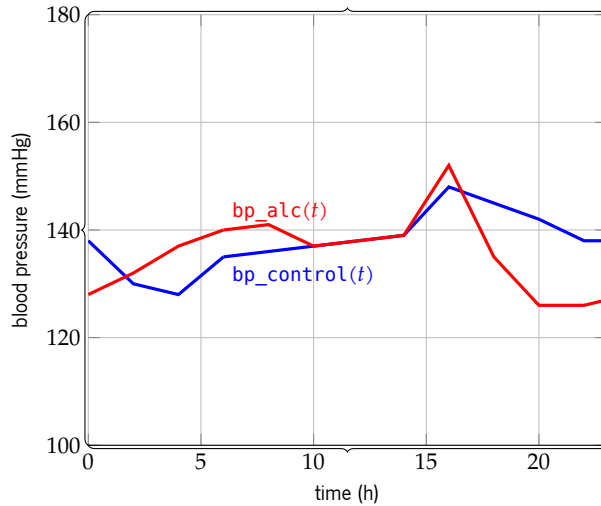


Figure 14: Blood pressure variation in a 24 hour period

It can be observed that the blood pressure is higher for the alcohol period in the first hours of the morning (from $0h$ until around $2h$), and lower in the period between the early evening (around $16h$) to late night ($24h$), being identical for the remaining hours. As the study concludes, the average blood pressure is equal for both groups. Suppose now that we intend to evaluate the predicate $\text{bp_control} \geq \text{bp_alc}$, which corresponds intuitively to evaluate "how greater" is the blood pressure for the control group, when comparing with the alcohol group. Analogously to the previous example, it is necessary to make a correspondence between the values of the diastolic blood pressure and the $[0, 1]$ interval, where values close to 0 correspond to low blood pressure and values close to 1 mean a high blood pressure. Consider w_0 as the state where variables bp_control and bp_alc are the functions represented in Figure 14. According to the semantics of Definition 7.0.2, and computing the values $r \in \mathbb{R}$ such that $\text{bp_control} \geq \text{bp_alc}$, we formalise such a computation as

$$\begin{aligned}
\llbracket \text{bp_control} \geq \text{bp_alc} \rrbracket &\stackrel{\text{def}}{=} \sum_{i \in I} \left\{ \prod_{j=1}^2 \llbracket t_j \rrbracket_{w_0}(r_j^i) \mid r_1^i \geq r_2^i \right\} \\
&= \sum_{i \in I} \left\{ \llbracket \text{bp_control} \rrbracket_{w_0}(r_1^i); \llbracket \text{bp_alc} \rrbracket_{w_0}(r_2^i) \mid r_1^i \geq r_2^i \right\} \\
&= \llbracket \text{bp_control} \rrbracket(0); \llbracket \text{bp_alc} \rrbracket_{w_0}(0) + \\
&\quad \dots + \llbracket \text{bp_control} \rrbracket_{w_0}\left(\frac{5}{3}\right); \llbracket \text{bp_alc} \rrbracket_{w_0}\left(\frac{5}{3}\right) \\
&\quad + \llbracket \text{bp_control} \rrbracket_{w_0}(134); \llbracket \text{bp_alc} \rrbracket_{w_0}(134) + \\
&\quad \dots + \llbracket \text{bp_control} \rrbracket_{w_0}(24); \llbracket \text{bp_alc} \rrbracket_{w_0}(24) \\
&= \max\{\min\{\llbracket \text{bp_control} \rrbracket(0), \llbracket \text{bp_alc} \rrbracket_{w_0}(0)\}, \\
&\quad \dots, \min\{\llbracket \text{bp_control} \rrbracket_{w_0}\left(\frac{5}{3}\right), \llbracket \text{bp_alc} \rrbracket_{w_0}\left(\frac{5}{3}\right)\}, \\
&\quad \min\{\llbracket \text{bp_control} \rrbracket_{w_0}\left(\frac{116}{7}\right), \llbracket \text{bp_alc} \rrbracket_{w_0}\left(\frac{116}{7}\right)\}, \\
&\quad \dots, \min\{\llbracket \text{bp_control} \rrbracket_{w_0}(24), \llbracket \text{bp_alc} \rrbracket_{w_0}(24)\}\} \\
&= \max\{\min\{\llbracket \text{bp_control} \rrbracket_{w_0}\left(\frac{116}{7}\right), \llbracket \text{bp_alc} \rrbracket_{w_0}\left(\frac{116}{7}\right)\}\} \\
&= \llbracket \text{bp_control} \rrbracket_{w_0}\left(\frac{116}{7}\right)
\end{aligned}$$

One of the solutions of $\text{bp_control}(t) = \text{bp_alc}(t)$ is $t = \frac{116}{7} \approx 16.5714$, which corresponds to the supremum of the values for which the predicate $\text{bp_control} \geq \text{bp_alc}$ holds. Hence, the truth degree of the predicate is $\llbracket \text{bp_control} \rrbracket_{w_0}\left(\frac{116}{7}\right)$.

Let us return to FAS and to the example below Definition 7.0.1, and consider the same state w_0 . Another example of a predicate is `02 is in 02_low`, whose interpretation in w_0 is the function

$$\llbracket \text{02 is in 02_low} \rrbracket_{w_0}(r) = \begin{cases} w_0(\text{02_low}, r) & \text{for the only } r \text{ such that } \llbracket \text{02} \rrbracket_{w_0}(r) = \top \\ \perp & \text{otherwise} \end{cases}$$

We are now able to introduce the semantics of atomic \mathcal{F}_2 -programs, that gives meaning to a weighted execution of a variable assignment.

Definition 7.0.3 (Interpretation of atomic programs). *Let $W : \mathbf{L}^{X^{\mathbb{R}}}$ be a set of states. The interpretation of atomic programs is a map*

$$\llbracket _ \rrbracket_0 : \text{Prog}_0 \rightarrow M^{\mathbf{L}}(W)$$

where $M^{\mathbf{L}}(W) = \{(w, \mathbf{L}^W)\}$, defined as

$$\llbracket x := t \rrbracket_0 = \{(w, \varphi(w, t)) \mid w \in W\}$$

where

$$\varphi(w)(t) = \lambda w' . \begin{cases} \top & \text{if } \forall x, x'. x' \neq x, r \in \mathbb{R} . w'(x')(r) = w(x')(r) \\ & \wedge w'(x)(r) = \llbracket t \rrbracket_w(r) \\ \perp & \text{otherwise} \end{cases}$$

Note that although the variables involved may be weighted, the assignment itself boils down to a binary multirelation on the set of states. Concretely, an *assignment* $x := t$ connects any state w with a set of states w' such that the weight of x in w' is the value of term t in w . In Example 7.0.1, the assignment $\llbracket \text{PIP_inc} := 5 \rrbracket$ attributes the term 5 to the variable `PIP_inc`, formally

$$\llbracket \text{PIP_inc} \rrbracket_{w_1}(r) = \llbracket 5 \rrbracket_{w_0}(r), \forall r \in \mathbb{R}$$

Defined the semantics of atomic programs, we are now in conditions to introduce the semantics of composition programming operators, given directly in terms of the operators \circ and \uplus on \mathbf{L} -valued weighted multirelations in $M^{\mathbf{L}}(W)$.

Definition 7.0.4 (Interpretation of non atomic programs). *Let W be a set of states. The interpretation of a program $\pi \in \text{Prog}$ is a map*

$$\llbracket _ \rrbracket : \text{Prog} \rightarrow M^{\mathbf{L}}(W)$$

recursively defined as

- $\llbracket \text{skip} \rrbracket = 1_{\circ}$.
- $\llbracket \pi_0 \rrbracket = \llbracket \pi_0 \rrbracket_0$, for each $\pi_0 \in \text{Prog}_0$
- $\llbracket \pi_1 ; \pi_2 \rrbracket = \llbracket \pi_1 \rrbracket \circ \llbracket \pi_2 \rrbracket$
- $\llbracket \pi_1 \parallel \pi_2 \rrbracket = \llbracket \pi_1 \rrbracket \uplus \llbracket \pi_2 \rrbracket$

How can this semantics be extended to include conditional statements? Actually, to capture **if-then-else** and **switch** statements requires the introduction of a suitable notion of a test. This will be defined in the next Chapter, since it relies on the satisfaction relation of the generated family of logics.

DYNAMIC LOGICS FOR WEIGHTED *MULTI-FLOW* COMPUTATIONS

We saw in Chapter 5 that the syntax and the semantics of the family of the generated dynamic logics for weighted “single-flow” computations, $\Gamma(\mathbf{L})$, is built over a non-empty set of variables X , terms over X and a data domain \mathbb{R} , for each complete right residuated lattice \mathbf{L} . We propose in this chapter a similar approach, for weighted “multi-flow” computations. Each complete right residuated lattice \mathbf{L} induces a $*$ -free dynamic logic to reason about \mathcal{F}_2 computations interpreted as weighted “multi-flow” computations. The result is a $*$ -free equational variant and a generalisation of *concurrent propositional dynamic logic (CPDL)* [Pe187]. On the one hand, its semantics include weights in the execution, and assertions to evaluate formulæ in a weighted truth space, as introduced in Chapter 7. On the other hand, instead of considering abstract programs and propositions, the family of logics is defined over a non-empty set of variables X , terms over X and a data domain \mathbb{R} .

We call this family of logics $\Omega(\mathbf{L})$. The signature, formulæ and satisfaction relation are presented below.

8.1 Generation of $*$ -free multi-valued equational dynamic logics

Once a language for programs is fixed (3), the set of formulæ of $\Omega(\mathbf{L})$ introduces, as expected, the universal and existential modalities over programs. Formally,

Definition 8.1.1. *A signature for $\Omega(\mathbf{L})$, built over a set of variables X , is a tuple*

$$\Delta = ((F, P), \Pi)$$

where (F, P) is a data signature composed by functional and predicate symbols and $\Pi \subseteq \text{Prog}_0$. The set of formulæ for Δ , denoted by $\text{Fm}^{\Omega(\mathbf{L})}(\Delta)$, is generated by

$$\rho ::= \top \mid \perp \mid p(t_1, \dots, t_n) \mid \rho \vee \rho \mid \rho \wedge \rho \mid \rho \rightarrow \rho \mid \langle \pi \rangle \rho$$

for $p(t_1, \dots, t_n) \in T_P(X)$ and $\pi \in \text{Prog}$.

Definition 8.1.2 (Satisfaction relation). *Let \mathbf{L} be a complete right residuated lattice, the graded satisfaction relation for $\Omega(\mathbf{L})$ consists of a function*

$$\vDash_{\Omega(\mathbf{L})} : W \times \text{Fm}^{\Omega(\mathbf{L})}(\Delta) \rightarrow \mathbf{L}$$

recursively defined by

- $(w \vDash_{\Omega(\mathbf{L})} \top) = \top$
- $(w \vDash_{\Omega(\mathbf{L})} \perp) = \perp$
- $(w \vDash_{\Omega(\mathbf{L})} p(t_1, \dots, t_n)) = \llbracket p(t_1, \dots, t_n) \rrbracket_w$
- $(w \vDash_{\Omega(\mathbf{L})} \rho \wedge \rho') = (w \vDash_{\Omega(\mathbf{L})} \rho) \cdot (w \vDash_{\Omega(\mathbf{L})} \rho')$
- $(w \vDash_{\Omega(\mathbf{L})} \rho \vee \rho') = (w \vDash_{\Omega(\mathbf{L})} \rho) + (w \vDash_{\Omega(\mathbf{L})} \rho')$
- $(w \vDash_{\Omega(\mathbf{L})} \rho \rightarrow \rho') = (w \vDash_{\Omega(\mathbf{L})} \rho) \rightarrow (w \vDash_{\Omega(\mathbf{L})} \rho')$
- $(w \vDash_{\Omega(\mathbf{L})} \langle \pi \rangle \rho) = \sum_{\varphi \in \pi_2(\llbracket \pi \rrbracket)} \left(\sum_{u \in W} (\varphi(u); (u \vDash_{\Omega(\mathbf{L})} \rho)) \right)$

The (graded) satisfaction of $(w \vDash_{\Omega(\mathbf{L})} \langle \pi \rangle \rho)$, in particular, is given by the weight of some state u on some weighted set φ which is related to state w by some weighted multirelation, and the weight of formula ρ on u .

Some properties of $\Omega(\mathbf{L})$

Now we establish some properties of $\Omega(\mathbf{L})$. The following two lemmas, the second one establishing auxiliary properties, are proved by using quasi-equational reasoning from the axiomatisation of \mathbf{L} , resorting to similar arguments to the proofs presented in reference [MNM16]. The remaining results resort to the semantics previously introduced.

Lemma 8.1.1. *Let \mathbf{L} be a complete \mathbb{I} -right residuated lattice. Then*

$$(w \vDash_{\Omega(\mathbf{L})} \rho \rightarrow \rho') = \top \text{ iff } (w \vDash_{\Omega(\mathbf{L})} \rho) \leq (w \vDash_{\Omega(\mathbf{L})} \rho') \quad (118)$$

$$(w \vDash_{\Omega(\mathbf{L})} \rho \leftrightarrow \rho') = \top \text{ iff } (w \vDash_{\Omega(\mathbf{L})} \rho) = (w \vDash_{\Omega(\mathbf{L})} \rho') \quad (119)$$

Lemma 8.1.2. *The following properties hold for any complete right residuated lattice \mathbf{L} :*

$$a \leq b \ \& \ c \leq d \ \Rightarrow \ a + c \leq b + d \quad (120)$$

$$a; (b \cdot c) \leq (a; b) \cdot (a; c) \quad (121)$$

For I finite, we also have

$$\sum_{i \in I} (a_i \cdot b_i) \leq \sum_{i \in I} a_i \cdot \sum_{i \in I} b_i \quad (122)$$

Lemma 8.1.3. *Let \mathbf{L} be a complete \mathbb{I} -right residuated lattice. The following are valid formulæ in any $\Omega(\mathbf{L})$:*

$$\langle \pi \rangle (\rho \vee \rho') \leftrightarrow \langle \pi \rangle \rho \vee \langle \pi \rangle \rho' \quad (123)$$

$$\langle \pi \rangle (\rho \wedge \rho') \rightarrow \langle \pi \rangle \rho \wedge \langle \pi \rangle \rho' \quad (124)$$

$$\langle \pi_1; \pi_2 \rangle \rho \leftrightarrow \langle \pi_1 \rangle \langle \pi_2 \rangle \rho \quad (125)$$

$$\langle \pi \rangle \perp \leftrightarrow \perp \quad (126)$$

$$\langle \pi_1 \| \pi_2 \rangle \rho \leftrightarrow \langle \pi_1 \rangle \rho \vee \langle \pi_2 \rangle \rho \quad (127)$$

Proof.

(123):

$$\begin{aligned}
& w \vDash_{\Omega(\mathbf{L})} \langle \pi \rangle (\rho \vee \rho') \\
= & \quad \{ \text{definition of } \vDash_{\Omega(\mathbf{L})} \} \\
& \sum_{\varphi \in \pi_2 \llbracket \pi \rrbracket} \left(\sum_{u \in W} (\varphi(u); (u \vDash_{\Omega(\mathbf{L})} \rho \vee \rho')) \right) \\
= & \quad \{ \text{definition of } \vDash_{\Omega(\mathbf{L})} \} \\
& \sum_{\varphi \in \pi_2 \llbracket \pi \rrbracket} \left(\sum_{u \in W} ((\varphi(u); ((u \vDash_{\Omega(\mathbf{L})} \rho) + (u \vDash_{\Omega(\mathbf{L})} \rho')))) \right) \\
= & \quad \{ (29) \} \\
& \sum_{\varphi \in \pi_2 \llbracket \pi \rrbracket} \left(\sum_{u \in W} (\varphi(u); (u \vDash_{\Omega(\mathbf{L})} \rho) + \varphi(u); (u \vDash_{\Omega(\mathbf{L})} \rho')) \right) \\
= & \quad \{ \text{by (25) and (26)} \} \\
& \sum_{\varphi \in \pi_2 \llbracket \pi \rrbracket} \left(\sum_{u \in W} (\varphi(u); (u \vDash_{\Omega(\mathbf{L})} \rho)) \right) + \sum_{\varphi \in \pi_2 \llbracket \pi \rrbracket} \left(\sum_{u \in W} (\varphi(u); (u \vDash_{\Omega(\mathbf{L})} \rho')) \right) \\
= & \quad \{ \text{definition of } \vDash_{\Omega(\mathbf{L})} \} \\
& (w \vDash_{\Omega(\mathbf{L})} \langle \pi \rangle \rho) + (w \vDash_{\Omega(\mathbf{L})} \langle \pi \rangle \rho') \\
= & \quad \{ \text{definition of } \vDash_{\Omega(\mathbf{L})} \} \\
& (w \vDash_{\Omega(\mathbf{L})} \langle \pi \rangle \rho \vee \langle \pi \rangle \rho')
\end{aligned}$$

Therefore, by (119), $\langle \pi \rangle (\rho \vee \rho') \leftrightarrow \langle \pi \rangle \rho \vee \langle \pi \rangle \rho'$ is valid.

(124):

$$\begin{aligned}
& (w \vDash_{\Omega(\mathbf{L})} \langle \pi \rangle (\rho \wedge \rho')) \\
= & \quad \{ \text{definition of } \vDash_{\Omega(\mathbf{L})} \} \\
& \sum_{\varphi \in \pi_2(\llbracket \pi \rrbracket)} \left(\sum_{u \in W} (\varphi(u); (u \vDash_{\Omega(\mathbf{L})} \rho \wedge \rho')) \right) \\
= & \quad \{ \text{definition of } \vDash_{\Omega(\mathbf{L})} \} \\
& \sum_{\varphi \in \pi_2(\llbracket \pi \rrbracket)} \left(\sum_{u \in W} (\varphi(u); ((u \vDash_{\Omega(\mathbf{L})} \rho) \cdot (w' \vDash_{\Omega(\mathbf{L})} \rho'))) \right) \\
\leq & \quad \{ \text{by (121) and (120)} \} \\
& \sum_{\varphi \in \pi_2(\llbracket \pi \rrbracket)} \left(\sum_{u \in W} ((\varphi(u); (u \vDash_{\Omega(\mathbf{L})} \rho)) \cdot (\varphi(u); (u \vDash_{\Omega(\mathbf{L})} \rho'))) \right) \\
\leq & \quad \{ \text{by (122)} \} \\
& \sum_{\varphi \in \pi_2(\llbracket \pi \rrbracket)} \left(\sum_{u \in W} (\varphi(u); (u \vDash_{\Omega(\mathbf{L})} \rho)) \right) \cdot \sum_{\varphi \in \pi_2(\llbracket \pi \rrbracket)} \left(\sum_{u \in W} (\varphi(u); (u \vDash_{\Omega(\mathbf{L})} \rho')) \right) \\
= & \quad \{ \text{definition of } \vDash_{\Omega(\mathbf{L})} \} \\
& (w \vDash_{\Omega(\mathbf{L})} \langle \pi \rangle \rho) \cdot (w \vDash_{\Omega(\mathbf{L})} \langle \pi \rangle \rho') \\
= & \quad \{ \text{definition of } \vDash_{\Omega(\mathbf{L})} \} \\
& (w \vDash_{\Omega(\mathbf{L})} \langle \pi \rangle \rho \wedge \langle \pi \rangle \rho')
\end{aligned}$$

Therefore, by (118), $\langle \pi \rangle (\rho \wedge \rho') \rightarrow \langle \pi \rangle \rho \wedge \langle \pi \rangle \rho'$ is valid.

(125):

$$\begin{aligned}
& (a, \varphi) \in \llbracket \pi_1; \pi_2 \rrbracket \\
\Leftrightarrow & \\
& \exists_{\varphi' \in \pi_2(\llbracket \pi_1 \rrbracket)} \wedge \exists_{F: W \rightarrow \mathbf{L}^W} \cdot \forall_{b \cdot (b, F(b)) \in \llbracket \pi_2 \rrbracket} \wedge \varphi(u) = \sum_{b \in W} (\varphi'(b); F(b)(u)), \forall_{u \in W}
\end{aligned}$$

Then, this yields

$$\begin{aligned}
& (w \vDash_{\Omega(\mathbf{L})} \langle \pi_1; \pi_2 \rangle \rho) \\
= & \quad \{ \text{definition of } \vDash_{\Omega(\mathbf{L})} \text{ and } \llbracket _ \rrbracket \} \\
& \sum_{u \in W} \left(\left(\sum_{b \in W} (\varphi'(b); F(b)(u)) \right); (u \vDash_{\Omega(\mathbf{L})} \rho) \right) \\
= & \quad \{ (26) \} \\
& \sum_{b \in W} \left(\left(\sum_{u \in W} (\varphi'(b); F(b)(u)) \right); (u \vDash_{\Omega(\mathbf{L})} \rho) \right) \\
= & \quad \{ (30) \} \\
& \sum_{b \in W} \left(\sum_{u \in W} (\varphi'(b); F(b)(u); (u \vDash_{\Omega(\mathbf{L})} \rho)) \right) \\
= & \quad \{ (29) \} \\
& \sum_{b \in W} \left(\varphi'(b); \left(\sum_{u \in W} (F(b)(u); (u \vDash_{\Omega(\mathbf{L})} \rho)) \right) \right)
\end{aligned}$$

But since $(w, \varphi') \in \pi_2(\llbracket \pi_1 \rrbracket)$ and $(b, F(b)) \in \llbracket \pi_2 \rrbracket$, $\forall b \in W$ we have

$$\sum_{b \in W} \left(\varphi'(b); \left(\sum_{u \in W} (F(b)(u); (u \vDash_{\Omega(\mathbf{L})} \rho)) \right) \right) = (w \vDash_{\Omega(\mathbf{L})} \langle \pi_1 \rangle \langle \pi_2 \rangle \rho)$$

Hence, by (119), $\langle \pi_1; \pi_2 \rangle \rho \leftrightarrow \langle \pi_1 \rangle \langle \pi_2 \rangle \rho$ is valid.

(126):

$$\begin{aligned}
& (w \vDash_{\Omega(\mathbf{L})} \langle \pi \rangle \perp) \\
= & \quad \{ \text{definition of } \vDash_{\Omega(\mathbf{L})} \} \\
& \sum_{\varphi | (w, \varphi) \in \llbracket \pi \rrbracket} \left(\sum_{u \in W} (\varphi(u); (u \vDash_{\Omega(\mathbf{L})} \perp)) \right) \\
= & \quad \{ \text{definition of } \vDash_{\Omega(\mathbf{L})} \} \\
& \sum_{\varphi | (w, \varphi) \in \llbracket \pi \rrbracket} \left(\sum_{u \in W} (\varphi(u); \perp) \right) \sum_{\varphi | (w, \varphi) \in \llbracket \pi \rrbracket} \left(\sum_{u \in W} \perp \right) \\
= & \quad \{ \text{by (40)} \} \\
& \perp
\end{aligned}$$

Therefore, by (119), $\langle \pi \rangle \perp \leftrightarrow \perp$ is valid. **(127):**

$$\begin{aligned}
& (w \vDash_{\Omega(\mathbf{L})} \langle \pi_1 \parallel \pi_2 \rangle \rho) \\
= & \quad \{ \text{definition of } \vDash_{\Omega(\mathbf{L})} \} \\
& \sum_{\varphi \in \pi_2(\llbracket \pi_1 \parallel \pi_2 \rrbracket)} \left(\sum_{u \in W} (\varphi(u); u \vDash_{\Omega(\mathbf{L})} \rho) \right) \\
= & \quad \{ \text{definition of } \llbracket _ \rrbracket \} \\
& \sum_{\varphi_1 \cup \varphi_2 \in \pi_2(\llbracket \pi_1 \parallel \pi_2 \rrbracket)} \left(\sum_{u \in W} ((\varphi_1 \cup \varphi_2)(u); (u \vDash_{\Omega(\mathbf{L})} \rho)) \right) \\
= & \quad \{ \text{definition of } \cup \text{ (3.4.1)} \} \\
& \sum_{\substack{\varphi_1 \in \pi_2(\llbracket \pi_1 \rrbracket) \\ \vee \varphi_2 \in \pi_2(\llbracket \pi_2 \rrbracket)}} \left(\sum_{u \in W} ((\varphi_1(u) + \varphi_2(u)); (u \vDash_{\Omega(\mathbf{L})} \rho)) \right) \\
= & \quad \{ (29) \} \\
& \sum_{\substack{\varphi_1 \in \pi_2(\llbracket \pi_1 \rrbracket) \\ \vee \varphi_2 \in \pi_2(\llbracket \pi_2 \rrbracket)}} \left(\sum_{u \in W} (\varphi_1(u); (u \vDash_{\Omega(\mathbf{L})} \rho) + \varphi_2(u); (u \vDash_{\Omega(\mathbf{L})} \rho)) \right) \\
= & \quad \{ (25), (26) \text{ and definition of } \parallel \} \\
& \sum_{\varphi_1 \in \pi_2(\llbracket \pi_1 \rrbracket)} \left(\sum_{u \in W} (\varphi_1(u); (u \vDash_{\Omega(\mathbf{L})} \rho)) \right) + \sum_{\varphi_2 \in \pi_2(\llbracket \pi_2 \rrbracket)} \left(\sum_{u \in W} (\varphi_2(u); (u \vDash_{\Omega(\mathbf{L})} \rho)) \right) \\
= & \quad \{ \text{definition of } \vDash_{\Omega(\mathbf{L})} \} \\
& w \vDash_{\Omega(\mathbf{L})} \langle \pi_1 \rangle \rho + w \vDash_{\Omega(\mathbf{L})} \langle \pi_2 \rangle \rho \\
= & \quad \{ \text{definition of } \vDash_{\Omega(\mathbf{L})} \} \\
& w \vDash_{\Omega(\mathbf{L})} \langle \pi_1 \rangle \rho \vee \langle \pi_2 \rangle \rho
\end{aligned}$$

Therefore, by (119), $\langle \pi_1 \parallel \pi_2 \rangle \rho \leftrightarrow \langle \pi_1 \rangle \rho \vee \langle \pi_2 \rangle \rho$ is valid. □

8.2 Back to the weighted *multi-flow* semantics

In order to capture conditional operators **if-then-else** and **switch**, we need a notion of test, whose syntax is, as mentioned in the previous chapter, $\rho?$, for a formula $\rho \in \text{Fm}^{\Omega(\mathbf{L})}(\Delta)$, for each signature Δ . Its interpretation is as follows:

test.

$$\llbracket \rho? \rrbracket = \left\{ (w, \varphi) \mid \varphi(w') = \begin{cases} w \vDash_{\Omega(\mathbf{L})} \rho & \text{if } w = w', w, w' \in W \\ \perp & \text{otherwise} \end{cases} \right\}$$

We can now give semantics to **if-then-else** statements, as follows:

Conditional.

$$\begin{aligned}
& \llbracket \mathbf{if} \ \rho \ \mathbf{then} \ \pi_1 \ \mathbf{else} \ \pi_2 \rrbracket \\
&= \llbracket \rho?; \pi_1 \parallel (\rho \rightarrow \perp)?; \pi_2 \rrbracket \\
&= \llbracket \rho? \rrbracket \circ \llbracket \pi_1 \rrbracket \uplus \llbracket (\rho \rightarrow \perp)? \rrbracket \circ \llbracket \pi_2 \rrbracket
\end{aligned}$$

The associativity of operator \uplus allows also to define a semantics for a generalised conditional, as follows.

Switch.

$$\begin{aligned}
& \llbracket \mathbf{switch} \ (\rho_i, \pi_i), \dots, (\rho_n, \pi_n) \rrbracket \\
&= \llbracket \rho_1?; \pi_1 \parallel \dots \parallel \rho_n?; \pi_n \rrbracket = \bigsqcup_i (\llbracket \rho_i? \rrbracket \circ \llbracket \pi_i \rrbracket)
\end{aligned}$$

Intuitively, both operators relate a state w to a weighted set of states w' , which assigns a weight to the execution of each π_i . Each one of these weights is the value of the evaluated predicate corresponding to the branch in which π_i is executed.

Although in standard imperative programming languages, the semantics of conditionals is expressed through choice (union), conditionals in FAS do not represent a choice, but a form of parallel evaluation. The parallel composition \uplus suits better the idea that we want to capture, since the execution of those commands in FAS do not represent a choice.

Consider the following simple example. Let $W = \{w_0, w_1, w_2\}$ be a set of states and consider weighted multirelations $R = \{(w_0, \varphi_1)\}$, with φ_1 such that $\varphi_1(w_1) = 0.4$, and $S = \{(w_0, \varphi_2)\}$, with φ_2 such that $\varphi_2(w_2) = 0.6$. The union of R and S is the set $\{(w_0, \varphi_1), (w_0, \varphi_2)\}$, which carries the standard interpretation of conditionals as a nondeterministic choice between (w_0, φ_1) and (w_0, φ_2) . On the other hand, the parallel composition $R \uplus S$ represents a single execution $\{(w_0, \varphi_1 \cup \varphi_2)\}$ from state w_0 going simultaneously to states w_1 and w_2 .

Let us now illustrate the suitability of the proposed semantics by computing the interpretation of the conditional fragment of our reference example (Example 7.0.1), choosing again $\mathbf{L} = \mathbf{G}$.

$$\begin{aligned}
& \llbracket \mathbf{if} \ 02 \ \mathbf{is} \ \mathbf{in} \ 02_low \ \mathbf{then} \ \text{PIP_inc:=5} \ \mathbf{else} \ \text{PIP_inc:=0} \ \mathbf{endif} \rrbracket \\
&= \llbracket p?; \text{PIP_inc:=5} \parallel (p \rightarrow \perp)?; \text{PIP_inc:=0} \rrbracket \\
&= \llbracket p? \rrbracket \circ \llbracket \text{PIP_inc:=5} \rrbracket \uplus \llbracket (p \rightarrow \perp)? \rrbracket \circ \llbracket \text{PIP_inc:=0} \rrbracket \\
&= \left\{ (w, \varphi_1) \mid \exists \varphi'_1 \cdot (w, \varphi'_1) \in \llbracket p? \rrbracket \wedge \exists_{F_1: W \rightarrow \mathbf{L}^W} \cdot \forall_{b \in W} \cdot (b, F_1(b)) \in \llbracket \text{PIP_inc:=5} \rrbracket \right. \\
&\quad \left. \wedge \varphi_1(u) = \sum_{b \in W} (\varphi'_1(b); F_1(b)(u)) \right\} \\
&\quad \uplus \left\{ (w, \varphi_2) \mid \exists \varphi'_2 \cdot (w, \varphi'_2) \in \llbracket (p \rightarrow \perp)? \rrbracket \wedge \exists_{F_2: W \rightarrow \mathbf{L}^W} \cdot \forall_{b \in W} \cdot (b, F_2(b)) \in \llbracket \text{PIP_inc:=0} \rrbracket \right. \\
&\quad \left. \wedge \varphi_2(u) = \sum_{b \in W} (\varphi'_2(b); F_2(b)(u)) \right\} \\
&= \left\{ (w, \varphi_1) \mid \varphi_1(u) = \varphi'_1(w); F_1(w)(u) \right\} \uplus \left\{ (w, \varphi_2) \mid \varphi_2(u) = \varphi'_2(w); F_2(w)(u) \right\}
\end{aligned}$$

$$\begin{aligned}
&= \{(w, \varphi_1) \mid \varphi_1(u) = \llbracket p \rrbracket_w; 1\} \uplus \{(w, \varphi_2) \mid \varphi_2(u) = \llbracket p \rightarrow \perp \rrbracket_w; 1\} \\
&= \{(w, \llbracket p \rrbracket_w)\} \uplus \{(w, \llbracket p \rightarrow \perp \rrbracket_w)\}
\end{aligned}$$

where $p \stackrel{abv}{=} 02$ is in `02_low`.

Introducing aggregation and defuzzification

Consider again the conditional statement of Example 7.0.1. Its execution can be finished considering two distinct possibilities:

- (A) The branches remain separated, and further instructions are executed in parallel. The information from different branches is taken into account by the user;
- (B) The information is combined, which results in a single output.

The semantics presented for the **if ρ then π_1 else π_2** statement captures the parallel behaviour of option (A). On the other hand, the execution of option (B) is described by **if ρ then π_1 else π_2 aggregate**. The command instructs the aggregation of the multiple variables obtained from the branching into a single variable. The resulting variable is computed in terms of the operators of the complete right residuated lattice \mathbf{L} , as follows:

aggregate

$$\llbracket \text{if } \rho \text{ then } \pi_1 \text{ else } \pi_2 \text{ aggregate} \rrbracket = \{(w, \varphi(w')) \mid w, w' \in W\}$$

where

$$\varphi(w') = \lambda w'. \begin{cases} \top & \text{if } \forall_{x', x' \neq x, r \in \mathbb{R}} \cdot w'(x')(r) = w(x')(r) \\ & \wedge w'(x)(r) = \sum_{u \in W} (\pi_2(\llbracket \rho ? \rrbracket)(u); u(x)(r)) + \sum_{u \in W} (\pi_2(\llbracket \rho \rightarrow \perp ? \rrbracket)(u); u(x)(r)) \\ \perp & \text{otherwise} \end{cases}$$

where w' is the “aggregated” state, i.e. the state where the multiple variables merge into one single variable x .

Intuitively, the command with aggregation represents a crisp execution, relating state w with the “aggregated” state w' . In such state the weight of x is calculated pointwise in terms of the weights of formulas ρ and $\rho \rightarrow \perp$, and of the weights of x in the branching states. The other variables declared in the program ($x' \neq x$) keep, of course, the same weights of state w after the execution of the controller.

Let us now illustrate the semantics of aggregation applied to our reference example, i.e. to the program

Example 8.2.1.

`02_low:=FUZZY SET((70,0),(75,1),(85,1),(90,0))`

`if 02 is in 02_low then PIP_inc:=5 else PIP_inc:=0
endif aggregate`

The semantics is given by

$$\begin{aligned} & \llbracket \text{if } 02 \text{ is in } 02_low \text{ then } PIP_inc:=5 \text{ else } PIP_inc:=0 \text{ endif aggregate} \rrbracket \\ & = \{(w, \varphi(w')) \mid w, w' \in W\} \end{aligned}$$

with

$$\varphi(w') = \begin{cases} \top & \text{if } w'(PIP_inc)(r) \\ = \sum_{u \in W} (\pi_2(\llbracket p? \rrbracket)(u); u(PIP_inc)(r)) + \sum_{u \in W} (\pi_2(\llbracket (p \rightarrow \perp)? \rrbracket)(u); u(PIP_inc)(r)) & \\ \perp & \text{otherwise} \end{cases}$$

where w' has the same meaning as before and $p \stackrel{abv}{=} 02 \text{ is in } 02_low$.

The variable PIP_inc in the aggregate state w' is defined as

$$\begin{aligned} w'(PIP_inc)(5) &= \pi_2(\llbracket p? \rrbracket)(u_1); u_1(PIP_inc)(5) + \pi_2(\llbracket (p \rightarrow \perp)? \rrbracket)(u_2); u_2(PIP_inc)(5) \\ &= \pi_2(\llbracket p? \rrbracket)(u_1); \top + \pi_2(\llbracket (p \rightarrow \perp)? \rrbracket)(u_2); \perp \\ &= \pi_2(\llbracket p? \rrbracket)(u_1) \end{aligned}$$

$$\begin{aligned} w'(PIP_inc)(0) &= \pi_2(\llbracket p? \rrbracket)(u_1); u_1(PIP_inc)(0) + \pi_2(\llbracket (p \rightarrow \perp)? \rrbracket)(u_2); u_2(PIP_inc)(0) \\ &= \pi_2(\llbracket p? \rrbracket)(u_1); \perp + \pi_2(\llbracket (p \rightarrow \perp)? \rrbracket)(u_2); \top \\ &= \pi_2(\llbracket (p \rightarrow \perp)? \rrbracket)(u_2) \end{aligned}$$

and \perp for other $r \in \mathbb{R}$

Note that the output variable, weighted by nature, carries uncertainty on the PIP_inc itself. Therefore, in order to obtain a crisp value from the execution of the fuzzy controller, an additional command needs

to be applied. Such a command is $y := \mathbf{defuzzify} x$, allowing to convert the variable x obtained from aggregation into a crisp output y . The literature offers a wide variety of methods to compute such a value, e.g. weighted average or center of gravity, among others. In this thesis, however, for illustration purposes, we chose the first option. For more details in this topic see reference [NW05]. Such a semantics is given by

defuzzify

$$\llbracket y := \mathbf{defuzzify} x \rrbracket = \{(w, \psi(w')) \mid w, w' \in W\}$$

where

$$\psi(w') = \lambda w'.$$

$$\left\{ \begin{array}{l} \top \quad \text{if } \forall_{y'. y' \neq y, r \in \mathbb{R}} \cdot w'(y')(r) = w(y')(r) \\ \quad \wedge w'(y)(r) = \begin{cases} \top & \text{if } r = \frac{\bigoplus_i (r_i \times w(x)(r_i))}{\bigoplus_i w(x)(r_i)}, i \in \mathbb{N} \\ \perp & \text{otherwise} \end{cases} \\ \perp \quad \text{otherwise} \end{array} \right.$$

where w is the state of the aggregated variable x , and \bigoplus is the arithmetic sum.

We interpret the **defuzzify** operator as a special type of assignment, which assigns to the weight of a variable y the weight of a variable x resulting from aggregation, and after applies the weighted average method for defuzzification.

Let us illustrate this semantics resorting again to Example 7.0.1, by applying the **defuzzify** operator.

Example 8.2.2.

```
02_low:=FUZZY SET((70,0),(75,1),(85,1),(90,0))
```

```
if 02 is in 02_low then PIP_inc:=5 else PIP_inc:=0
```

```
endif aggregate;
```

```
def_PIP:=defuzzified PIP_inc
```

The semantics is computed as follows

$\llbracket \text{if } 02 \text{ is in } 02_low \text{ then } PIP_inc := 5 \text{ else } PIP_inc := 0 \text{ endif aggregate};$
 $\text{def_PIP} := \text{defuzzified } PIP_inc \rrbracket$
 $= \{(w, \psi(w'')) \mid w, w'' \in W\}$

with

$$\psi(w'') = \begin{cases} \top & \text{if } w''(\text{def_PIP})(r) = \begin{cases} \top & \text{if } r = \frac{5 \times w'(PIP_inc)(5) \oplus 0 \oplus \dots \oplus 0 \oplus 0 \times w'(PIP_inc)(0)}{w'(PIP_inc)(5) \oplus 0 \oplus \dots \oplus 0 \oplus w'(PIP_inc)(0)}, w' \in W \\ \perp & \text{otherwise} \end{cases} \\ \perp & \text{otherwise} \end{cases}$$

where $w' \in W$ is the state of the aggregated variable PIP_inc . Hence the variable def_PIP in state w'' is defined as

$$w''(\text{def_PIP})(r) = \begin{cases} \top & \text{if } r = \frac{5 \times w'(PIP_inc)(5)}{w'(PIP_inc)(5) \oplus w'(PIP_inc)(0)}, w' \in W \\ \perp & \text{otherwise} \end{cases}$$

Lemma 8.2.1. *Let \mathbf{L} be a complete \mathbb{H} -right residuated lattice. The following formula is valid in any $\Omega(\mathbf{L})$.*

$$\langle \rho? \rangle \rho' \leftrightarrow (\rho \wedge \rho')$$

Proof.

$$\begin{aligned} & (w \vDash_{\Omega(\mathbf{L})} \langle \rho? \rangle \rho') \\ = & \quad \{ \text{definition of } \vDash_{\Omega(\mathbf{L})} \} \\ & \sum_{\varphi \in \pi_2(\llbracket \rho? \rrbracket)} \left(\sum_{u \in W} (\varphi(u); u \vDash_{\Omega(\mathbf{L})} \rho') \right) \\ = & \quad \{ \text{definition of } \llbracket _ \rrbracket \} \\ & (w \vDash_{\Omega(\mathbf{L})} \rho); (w \vDash_{\Omega(\mathbf{L})} \rho') \\ = & \quad \{ \mathbf{L} \text{ is } \mathbb{H} \} \\ & (w \vDash_{\Omega(\mathbf{L})} \rho) \wedge (w \vDash_{\Omega(\mathbf{L})} \rho') \\ = & \quad \{ \text{definition of } \vDash_{\Omega(\mathbf{L})} \} \\ & (w \vDash_{\Omega(\mathbf{L})} \rho \wedge \rho') \end{aligned}$$

Hence, by (119), $(w \vDash_{\Omega(\mathbf{L})} \langle \rho? \rangle \rho') \leftrightarrow (w \vDash_{\Omega(\mathbf{L})} \rho \wedge \rho')$. □

8.3 An illustration with fuzzy arden syntax

To illustrate the application of our framework let us resort to our reference example 7.0.1, i.e. the conditional

$$\pi \stackrel{abv}{=} \text{if } O_2 \text{ is in } O_2_low \text{ then PIP_inc:=5 else PIP_inc:=0}$$

Let us also assume that, in an initial state w_0 , the patient has a O_2 level of 89¹. Our goal is to verify how ‘normal’ the O_2 level of the patient becomes after running the controller from state w_0 . Formally, this problem corresponds to computing the truth degree of the following $\Omega(\mathbf{L})$ formula

$$w_0 \vDash_{\Omega(\mathbf{L})} \langle \pi \rangle \rho$$

where $\rho \stackrel{abv}{=} O_2 \text{ is in } O_2_normal$.

In order to shorten the proof, we use the encoding defined in Chapter 7 to represent program π as

$$p?; \pi_1 \parallel (p \rightarrow \perp)?; \pi_2$$

where

$$\pi_1 \stackrel{abv}{=} \text{PIP_inc:=5}$$

$$\pi_2 \stackrel{abv}{=} \text{PIP_inc:=0}$$

$$p \stackrel{abv}{=} O_2 \text{ is in } O_2_low$$

The weight of

$$w_0 \vDash_{\Omega(\mathbf{L})} \langle p?; \pi_1 \parallel (p \rightarrow \perp)?; \pi_2 \rangle \rho \tag{128}$$

is obtained from the satisfaction relation established in Definition 8.1.2, as follows².

-
- 1 Note that the system only suggests a modification to the value of PIP_inc and this always depends on the manual action of some health professional.
 - 2 Although the lattice chosen to instantiate the problem was the \mathbf{G} lattice, and thus $;\equiv \min$, the semantics is the same for the other instantiations and \mathbf{P} , since $1;I = I$, for all $I \in \mathbf{L}$

$$\begin{aligned}
& w_0 \vDash_{\Omega(\mathbf{L})} \langle p?; \pi_1 \parallel (p \rightarrow \perp)?; \pi_2 \rangle \rho \\
= & \quad \{ \text{Lemma 8.1.3} \} \\
& w_0 \vDash_{\Omega(\mathbf{L})} \langle p? \rangle \langle \pi_1 \rangle \rho + w_0 \vDash_{\Omega(\mathbf{L})} \langle (p \rightarrow \perp)? \rangle \langle \pi_2 \rangle \rho \\
= & \quad \{ \text{definition of } \vDash_{\Omega(\mathbf{L})} \} \\
& \sum_{\psi_1 \in \pi_2(\llbracket p? \rrbracket)} \left(\sum_{u \in W} \psi_1(u); \left(\sum_{\varphi_1 \in \pi_2(\llbracket \pi_1 \rrbracket)} \left(\sum_{v \in W} \varphi_1(v); v \vDash_{\Omega(\mathbf{L})} \rho \right) \right) \right) \\
& + \sum_{\psi_2 \in \pi_2(\llbracket (p \rightarrow \perp)? \rrbracket)} \left(\sum_{u \in W} \psi_2(u); \left(\sum_{\varphi_2 \in \pi_2(\llbracket \pi_2 \rrbracket)} \left(\sum_{v \in W} \varphi_2(v); v \vDash_{\Omega(\mathbf{L})} \rho \right) \right) \right) \\
= & \quad \{ \text{definition of } \llbracket _ \rrbracket \} \\
& (w_0 \vDash_{\Omega(\mathbf{L})} p); (\varphi_1(v_1); v_1 \vDash_{\Omega(\mathbf{L})} \rho) + (w_0 \vDash_{\Omega(\mathbf{L})} p \rightarrow \perp); (\varphi_2(v_2); v_2 \vDash_{\Omega(\mathbf{L})} \rho) \\
= & \quad \{ \text{definitions of } \llbracket _ \rrbracket \} \\
& (w_0 \vDash_{\Omega(\mathbf{L})} p); (\top; v_1 \vDash_{\Omega(\mathbf{L})} \rho) + (w_0 \vDash_{\Omega(\mathbf{L})} p \rightarrow \perp); (\top; v_2 \vDash_{\Omega(\mathbf{L})} \rho) \\
= & \quad \{ (28) \} \\
& (w_0 \vDash_{\Omega(\mathbf{L})} p); (v_1 \vDash_{\Omega(\mathbf{L})} \rho) + (w_0 \vDash_{\Omega(\mathbf{L})} p \rightarrow \perp); (v_2 \vDash_{\Omega(\mathbf{L})} \rho)
\end{aligned}$$

In several situations, as the one of Example 7.0.1, some contextual assumptions must be taken into account. Thus, to verify a formula in $\Omega(\mathbf{L})$, we assume the validity of a set of contextual rules Γ . Notation

$$(\Gamma, w \vDash_{\Omega(\mathbf{L})} \rho)$$

stands for the value of $(w \vDash_{\Omega(\mathbf{L})} \rho)$ assuming Γ , i.e. given that $(w \vDash_{\Omega(\mathbf{L})} \Gamma) = \top$.

In order to interpret our problem, we use a concrete evaluation environment encoded in the following context rules

$$\Gamma = \begin{cases} \text{PIP_inc} = 0 \rightarrow \text{O}_2 = 89 \\ \text{PIP_inc} = 5 \rightarrow \text{O}_2 = 92 \end{cases}$$

which describes the impact of acting on the ventilator's PIP_inc, on the O_2 level of the patient. The increment of PIP_inc by 5 units results in an increase of the O_2 from 89 to 92, whereas if no action is performed, O_2 does not suffer any modification. In turn, these new values for O_2 entail a new weight for the predicate (e.g. `O2 is in O2_normal`).

It is relevant to mention that the output of the program, which may be according to option (A) or option (B), and the concrete decision of the health professional operating the system do not necessarily coincide. The output of option (A), i.e. the function $\llbracket \text{O2_low} \rrbracket_w + \llbracket \text{O2_low} \rrbracket_w \rightarrow 0$ (see Section 8.1), highlights which branch is more relevant to execute in the concrete scenario. This way, if no aggregation is performed, we are in presence of an uncertainty about which path will be chosen by the user. It is precisely over such an uncertainty that the logic reasons about. The actual execution of the command, which will be decided according to the weight of each branch, is not reflected in the logic.

Assuming Γ we can discuss the problem instantiating lattice \mathbf{L} with different complete right residuated lattices, namely \mathbf{L} , \mathbf{G} and \mathbf{P} . Starting with \mathbf{L} , we obtain

$$0.2;1 + 0.8;0.8 = \max\{0.2, 0.6\} = 0.6$$

The value 0.6 is interpreted as the truth degree that the execution of the fuzzy controller π adjusts the O_2 level of the patient to 'normal'. The formula (128) evaluates precisely the impact of the suggestion made by the fuzzy controller on the O_2 level of the patient. Note, in particular, that the weights of the predicates `O2 is in O2_low`, `O2 is in O2_low \rightarrow 0` and `O2 is in O2_normal` affect the value of the formula, with the 'else' branch having a stronger impact due to its higher weight compared to the 'then' branch. Considering now the lattice \mathbf{G} , it yields

$$\max\{\min\{0.2, 1\}, \min\{0, 0.8\}\} = \max\{0.2, 0\} = 0.2$$

Finally, the instantiation with \mathbf{P} entails

$$\max\{0.2 \times 1, 0 \times 0.8\} = \max\{0.2, 0\} = 0.2$$

Despite the strictness of the logic, the discrepancy between the values of the three lattices suggests that the truth space for each concrete application context needs to be carefully selected by the user.

We have seen that **aggregate** combines the multiple variables, obtained from the diverse branches, into a single variable. Therefore the application of the logic to a program with aggregation produces, in practice, the same result of the program without aggregation, i.e. it reasons about which branch of the fuzzy controller is more relevant to execute.

To illustrate the application of the logic to a program with **defuzzify**, consider the following program

```
 $\pi'' \stackrel{abv}{=} \text{if } O2 \text{ is in } O2\_low \text{ then } PIP\_inc:=5 \text{ else } PIP\_inc:=0$ 
  endif aggregate;
  def\_PIP:= defuzzified PIP\_inc
```

The goal is to verify how much the O_2 level of the patient becomes 'normal' after running such a controller, from state w_0 . The $\Omega(\mathbf{L})$ formula which corresponds to this problem is

$$w_0 \models_{\Omega(\mathbf{L})} \langle \pi'' \rangle \rho$$

considering the sequential composition of program

```
 $\pi' \stackrel{abv}{=} \text{if } O2 \text{ is in } O2\_low \text{ then } PIP\_inc:=5 \text{ else } PIP\_inc:=0$ 
```


endif aggregate

with assignment $\pi_3 \stackrel{abv}{=} \text{def_PIP} := \text{defuzzified PIP_inc}$. Let us use again the encoding of Chapter 7 to abbreviate program π'' as

$$\pi'; \pi_3$$

Thus the weight of

$$w_0 \vDash_{\Omega(\mathbf{L})} \langle \pi'; \pi_3 \rangle \rho \quad (129)$$

is calculated through the satisfaction relation of Definition 8.1.2 and by interpreting the program with defuzzification as in Example 8.2.2, as follows.

$$\begin{aligned} & w_0 \vDash_{\Omega(\mathbf{L})} \langle \pi'; \pi_3 \rangle \rho \\ = & \quad \{ (125) \} \\ & w_0 \vDash_{\Omega(\mathbf{L})} \langle \pi' \rangle \langle \pi_3 \rangle \rho \\ = & \quad \{ \text{definition of } \vDash_{\Omega(\mathbf{L})} \} \\ & \sum_{u \in W} \left(\pi_2(\llbracket \pi' \rrbracket)(w_0, u); \left(\sum_{v \in W} (\pi_2(\llbracket \pi_3 \rrbracket)(u, v); v \vDash_{\Omega(\mathbf{L})} \rho) \right) \right) \\ = & \quad \{ \text{definition of } \llbracket _ \rrbracket \} \\ & \sum_{u \in W} \left(\varphi(u); \left(\sum_{v \in W} (\psi(v); v \vDash_{\Omega(\mathbf{L})} \rho) \right) \right) \end{aligned}$$

$$\text{such that } \varphi(u) = \begin{cases} \top & \text{if } u(\text{PIP_inc})(r) = \begin{cases} \llbracket p \rrbracket_{w_0} & \text{if } r = 5 \\ \llbracket p \rightarrow \perp \rrbracket_{w_0} & \text{if } r = 0 \\ \perp & \text{otherwise} \end{cases} \\ \perp & \text{otherwise} \end{cases}$$

$$\text{and } \psi(v) = \begin{cases} \top & \text{if } v(\text{def_PIP})(r) = \begin{cases} \top & \text{if } r = \frac{5 \times u(\text{PIP_inc})(5) \oplus 0 \times u(\text{PIP_inc})(0)}{u(\text{PIP_inc})(5) \oplus u(\text{PIP_inc})(0)} \\ \perp & \text{otherwise} \end{cases} \\ \perp & \text{otherwise} \end{cases}$$

where \oplus is the arithmetic sum.

Considering the same values for O_2 , and the context rules Γ , and instantiating with the usual three lattices, the value of $v(\text{def_PIP})(r)$, i.e. the application of command `defuzzify` to variable `PIP_inc` is given by

$$\mathbf{t} : 1 \text{ for } r = \frac{5 \times 0.2 + 0 \times 0.8}{0.2 + 0.8} = \frac{1}{1} = 1 \quad \text{and } 0 \text{ otherwise}$$

$$\mathbf{G} : 1 \text{ for } r = \frac{5 \times 0.2 + 0 \times 0}{0.2 + 0} = \frac{1}{0.2} = 5 \quad \text{and 0 otherwise}$$

$$\mathbf{P} : 1 \text{ for } r = \frac{5 \times 0.2 + 0 \times 0}{0.2 + 0} = \frac{1}{0.2} = 5 \quad \text{and 0 otherwise}$$

Let us consider now the additional context rule $\text{def_PIP} = 1 \rightarrow O_2 = 89.5$, which means that the increment of PIP by 1 increases O_2 from 89 in state w_0 to 89.5 in the ‘defuzzified’ state v . Note that for $\text{def_PIP} = 5$ we already assumed the rule $\text{PIP} = 5 \rightarrow O_2 = 92$. The final step is to compute the weights of formula 8.3 using lattices \mathbf{L} , \mathbf{G} and \mathbf{P} , in function of the calculated average for each lattice.

$$\mathbf{L} : v(\text{def_PIP})(1); \llbracket \rho \rrbracket_u = 1; 0.9 = \max\{0, 1 + 0.9 - 1\} = 0.9$$

$$\mathbf{G} : v(\text{def_PIP})(5); \llbracket \rho \rrbracket_u = \min\{1, 1\} = 1$$

$$\mathbf{P} : v(\text{def_PIP})(5); \llbracket \rho \rrbracket_u = 1 \times 1 = 1$$

The interpretation of this example with defuzzification is that the logic formula reflects the effect that the suggestion of increasing the PIP of the patient by 1 and 5 has in the O_2 level. Note that, in particular for lattices \mathbf{G} and \mathbf{P} , the weight 1 means that, by running the controller and incrementing PIP by the suggested value, the O_2 level of the patient becomes ‘completely normal’.

8.4 An illustration with jFuzzyLogic

In this section we provide another application of our framework to an instance of the language \mathcal{F}_2 , called jFuzzyLogic [CAf13]. The language is an open source Java library, offering a complete implementation of fuzzy inference systems. It comes with a programming interface and a plugin for the Eclipse IDE to write and test code for fuzzy applications. Although jFuzzyLogic does not introduce any new feature relatively to FAS, we intend to illustrate the versatility of the introduced semantics and dynamic logics in capturing distinct application scenarios.

We illustrate the application of the logic with the following example, obtained from documentation of jFuzzyLogic [jfu], of a fuzzy controller for a container crane.

Example 8.4.1.

```
VAR_INPUT
distance: REAL;
angle: REAL;
END_VAR
```

```
VAR_OUTPUT
power: REAL
END_VAR
```

```

FUZZIFY distance
TERM too_far:=(0,1) (55,1) (0,0);
TERM zero:=(-5,0) (0,1) (5,0);
TERM close:=(0,0) (5,1) (10,0);
TERM medium:=(5,0) (10,1) (22,0);
TERM far:=(10,0) (22,1) (30,1);
END_FUZZIFY

```

```

FUZZIFY angle
TERM neg_big:=(-90,1) (-50,1) (-5,0);
TERM neg_small:=(-50,0) (-5,1) (0,0);
TERM zero:=(-5,0) (0,0) (5,0);
TERM pos_small:=(0,0) (5,1) (50,0);
TERM pos_big:=(5,0) (50,1) (90,1);
END_FUZZIFY

```

```

DEFUZZIFY power
TERM neg_high:=-26;
TERM neg_medium:=-8;
TERM zero:=0;
TERM pos_medium:=8;
TERM pos_high:=26;
END_DEFUZZIFY

```

```

RULEBLOCK
Rule1: IF distance IS medium AND angle IS pos_small
THEN power IS pos_medium
Rule2: IF distance IS medium AND angle IS zero THEN power IS zero
Rule3: IF distance IS far AND angle IS zero THEN power IS pos_medium
END_RULEBLOCK
END_FUNCTIONBLOCK

```

The system automates the control of the crane, by calculating the motor power to be applied, in function of the crane head position (distance) and the angle of the container sway (angle).

Analogously to FAS, the program above is divided into distinct blocks: VAR_INPUT and VAR_OUTPUT declare the input and output variables, respectively; the FUZZIFY and DEFUZZIFY blocks define a list of weighted sets for each one of those variables. For example, the statement

TERM close:= (0,0), (5,1), (10,0) defines variable close, in a state w , as the weighted set

$$\llbracket \text{close} \rrbracket_w(r) = w(\text{close}, r) = \begin{cases} \frac{r}{5} - 1 & \text{if } 5 \leq r \leq 10 \\ \frac{r}{12} + \frac{11}{6} & \text{if } 10 \leq r \leq 22 \end{cases}$$

which assigns, to each (crisp) value of distance, a value in the real interval $[0, 1]$ to evaluate how ‘close’ the container is to the target position. Figures 15 and 16 are the graphical representations of the variable `close`, and the remaining weighted sets defined in program of Example 8.4.1.

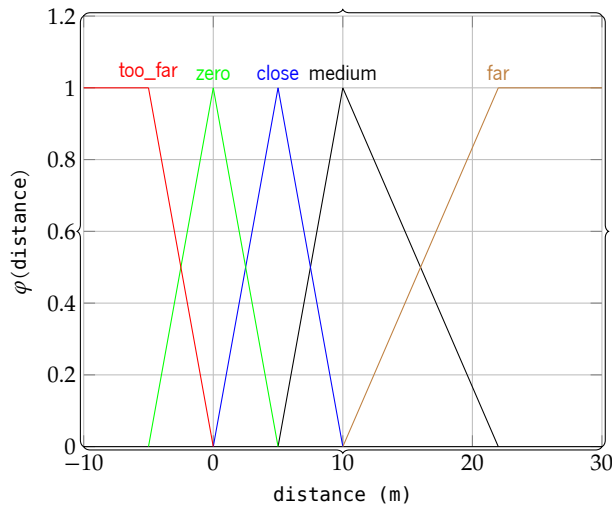


Figure 15: Graph distance between crane head and target position

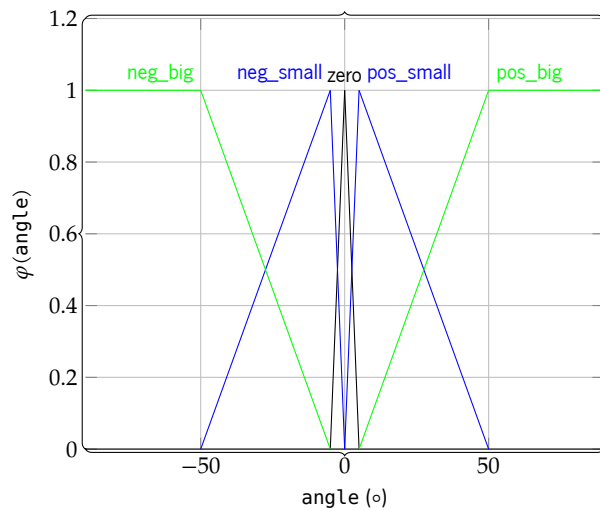


Figure 16: Graph angle of the container to the crane head

Note that variable power is defined as a constant, e.g. as

$$\llbracket \text{pos_medium} \rrbracket_w(r) = \begin{cases} \top & \text{if } r = 8 \\ \perp & \text{otherwise} \end{cases}$$

Finally, the RULEBLOCK sets the fuzzy controller itself, i.e. the **if-then** rules which automate the control of the container crane. In jFuzzyLogic, the logical connectives such as AND, and the ‘aggregation’ and ‘defuzzification’ methods are defined in the RULEBLOCK and the DEFUZZIFY block of the output variable, respectively. Since our framework is parametric on a right residuated lattice, those operators and methods will depend on the concrete lattice chosen. Analogously to previous examples, we will instantiate with the three usual action lattices **L**, **G** and **P**.

Let us consider an initial state w_0 where the distance between the crane and the target is $12m$ and the angle of the container to the crane head is 4° . We want to verify the certainty that, after running controller from w_0 , the container is ‘close’ to the target. To simplify the example, we opt to analyse only the distance, not including the angle in the predicate to be evaluated. This corresponds to compute the $\Omega(\mathbf{L})$ formula

$$w_0 \vDash_{\Omega(\mathbf{L})} \langle \pi \rangle \rho$$

where π is the program in the RULEBLOCK and $\rho \stackrel{abv}{=} \text{distance IS close}$.

In order to shorten the proof, we encode program π , resorting again to syntax given in Chapter 7, as

$$(p_1 \wedge p_3)?; q_1? \parallel (p_1 \wedge p_4)?; q_2? \parallel (p_2 \wedge p_4)?; q_1?$$

where

$$p_1 \stackrel{abv}{=} \text{distance IS medium}$$

$$p_2 \stackrel{abv}{=} \text{distance is far}$$

$$p_3 \stackrel{abv}{=} \text{angle IS pos_small}$$

$$p_4 \stackrel{abv}{=} \text{angle IS zero}$$

$$q_1 \stackrel{abv}{=} \text{power IS pos_medium}$$

$$q_2 \stackrel{abv}{=} \text{power IS zero}$$

The weight of

$$w_0 \vDash_{\Omega(\mathbf{L})} \langle (p_1 \wedge p_3)?; q_1? \parallel (p_1 \wedge p_4)?; q_2? \parallel (p_2 \wedge p_4)?; q_1? \rangle \rho \quad (130)$$

is obtained through the satisfaction relation 8.1.2, as follows:

$$\begin{aligned}
& w_0 \vDash_{\Omega(\mathbf{L})} \langle (p_1 \wedge p_3)?; q_1? \parallel (p_1 \wedge p_4)?; q_2? \parallel (p_2 \wedge p_4)?; q_1? \rangle \rho \\
= & \quad \{ (127) \} \\
& w_0 \vDash_{\Omega(\mathbf{L})} \langle (p_1 \wedge p_3)? \rangle \langle q_1? \rangle \rho \\
& + w_0 \vDash_{\Omega(\mathbf{L})} \langle (p_1 \wedge p_4)? \rangle \langle q_2? \rangle \rho \\
& + w_0 \vDash_{\Omega(\mathbf{L})} \langle (p_2 \wedge p_4)? \rangle \langle q_1? \rangle \rho \\
= & \quad \{ \text{definition of } \vDash_{\Omega(\mathbf{L})} \} \\
& \sum_{\psi_1 \in \pi_2(\llbracket (p_1 \wedge p_3)? \rrbracket)} \left(\sum_{u \in W} (\psi_1(u); \sum_{\varphi_1 \in \pi_2(\llbracket q_1? \rrbracket)} \left(\sum_{v \in W} \varphi_1(v); v \vDash_{\Omega(\mathbf{L})} \rho \right)) \right) \\
& + \sum_{\psi_2 \in \pi_2(\llbracket (p_1 \wedge p_4)? \rrbracket)} \left(\sum_{u \in W} (\psi_2(u); \sum_{\varphi_2 \in \pi_2(\llbracket q_2? \rrbracket)} \left(\sum_{v \in W} \varphi_2(v); v \vDash_{\Omega(\mathbf{L})} \rho \right)) \right) \\
& + \sum_{\psi_3 \in \pi_2(\llbracket (p_2 \wedge p_4)? \rrbracket)} \left(\sum_{u \in W} (\psi_3(u); \sum_{\varphi_1 \in \pi_2(\llbracket q_1? \rrbracket)} \left(\sum_{v \in W} \varphi_1(v); v \vDash_{\Omega(\mathbf{L})} \rho \right)) \right) \\
= & \quad \{ \text{definition of } \llbracket _ \rrbracket \} \\
& (w_0 \vDash_{\Omega(\mathbf{L})} p_1 \wedge p_3); (w_0 \vDash_{\Omega(\mathbf{L})} q_1); (w_0 \vDash_{\Omega(\mathbf{L})} \rho) \\
& + (w_0 \vDash_{\Omega(\mathbf{L})} p_1 \wedge p_4); (w_0 \vDash_{\Omega(\mathbf{L})} q_2); (w_0 \vDash_{\Omega(\mathbf{L})} \rho) \\
& + (w_0 \vDash_{\Omega(\mathbf{L})} p_2 \wedge p_4); (w_0 \vDash_{\Omega(\mathbf{L})} q_1); (w_0 \vDash_{\Omega(\mathbf{L})} \rho) \\
= & \quad \{ \text{definition of } \vDash_{\Omega(\mathbf{L})} \} \\
& ((w_0 \vDash_{\Omega(\mathbf{L})} p_1) \cdot (w_0 \vDash_{\Omega(\mathbf{L})} p_3)); (w_0 \vDash_{\Omega(\mathbf{L})} q_1); (w_0 \vDash_{\Omega(\mathbf{L})} \rho) \\
& + ((w_0 \vDash_{\Omega(\mathbf{L})} p_1) \cdot (w_0 \vDash_{\Omega(\mathbf{L})} p_4)); (w_0 \vDash_{\Omega(\mathbf{L})} q_2); (w_0 \vDash_{\Omega(\mathbf{L})} \rho) \\
& + ((w_0 \vDash_{\Omega(\mathbf{L})} p_2) \cdot (w_0 \vDash_{\Omega(\mathbf{L})} p_4)); (w_0 \vDash_{\Omega(\mathbf{L})} q_1); (w_0 \vDash_{\Omega(\mathbf{L})} \rho)
\end{aligned}$$

Analogously to the illustration in FAS we assume a set of contextual rules to reason about logical properties.

$$\Gamma = \begin{cases} \text{power IS pos_medium} \rightarrow \text{distance} = 7 \\ \text{power IS zero} \rightarrow \text{distance} = 12 \end{cases}$$

These rules model the impact that the motor power has in the position of the crane. If no action is performed (`power IS zero`), the distance to the target is maintained. If a motor power of 8 kW is applied (`power IS pos_medium`), at a certain instance, the distance is reduced to 7 yards. We then evaluate predicate `distance IS close` in the ‘aggregated’ state. Instantiating with the three habitual lattices, the weight of formula (130) is computed as

$$\begin{aligned}
\mathbf{L} & : (0.9 \cdot 0.8); 1; 0.6 + (0.9 \cdot 0.2); 1; 0 + (0.1 \cdot 0.2); 1; 0.6 \\
& = \max\{\max\{0, \min\{0.9, 0.8\} + 0.6 - 1\}, 0, \max\{0, \min\{0.1, 0.2\} + 0.6 - 1\}\} \\
& = \max\{0.4, 0, 0\}
\end{aligned}$$

$$= 0.4$$

$$\mathbf{G} : \max\{\min\{\min\{0.9, 0.8\}, 0.6\}, \min\{\min\{0.9, 0.2\}, 0\}, \min\{\min\{0.1, 0.2\}, 0.6\}\}$$

$$= \max\{0.6, 0, 0.1\}$$

$$= 0.6$$

$$\mathbf{P} : \max\{\min\{0.9, 0.8\} \times 0.6, \min\{0.9, 0.2\} \times 0, \min\{0.1, 0.2\} \times 0.6\}$$

$$= \max\{0.48, 0, 0.06\}$$

$$= 0.48$$

These values represent the truth degree that the execution of the controller reduces the distance of the container to the target from 12 to 5 yards.

We now exemplify the application of the logic to the program with defuzzification

```

 $\pi'' \stackrel{abv}{=} \text{IF distance IS medium AND angle IS pos\_small THEN power IS pos\_medium}$ 
 $\text{IF distance IS medium AND angle IS zero THEN power IS zero}$ 
 $\text{IF distance IS far AND angle IS zero THEN power IS pos\_medium}$ 
aggregate;
power_def := defuzzified power

```

We compute the weight corresponding to how 'close' the container crane is to the target position after running the controller, which corresponds to the $\Omega(\mathbf{L})$ formula

$$w_0 \models_{\Omega(\mathbf{L})} \langle \pi'' \rangle \rho$$

where $\pi'' = \pi'; \pi_0$ is the sequential composition of program

```

 $\pi' \stackrel{abv}{=} \text{IF distance IS medium AND angle IS pos\_small THEN power IS pos\_medium}$ 
 $\text{IF distance IS medium AND angle IS zero THEN power IS zero}$ 
 $\text{IF distance IS far AND angle IS zero THEN power IS pos\_medium}$ 
aggregate

```

with assignment $\pi_0 \stackrel{abv}{=} \text{power_def := defuzzified power.}$

The weight of

$$w_0 \models_{\Omega(\mathbf{L})} \langle \pi'; \pi_0 \rangle \rho \tag{131}$$

is computed as

$$\begin{aligned}
& w_0 \vDash_{\Omega(\mathbf{L})} \langle \pi'; \pi_0 \rangle \rho \\
= & \quad \{ (125) \} \\
& w_0 \vDash_{\Omega(\mathbf{L})} \langle \pi' \rangle \langle \pi_0 \rangle \rho \\
= & \quad \{ \text{definition of } \vDash_{\Omega(\mathbf{L})} \} \\
& \sum_{u \in W} \left(\pi_2(\llbracket \pi' \rrbracket)(w_0, u); \left(\sum_{v \in W} (\pi_2(\llbracket \pi_0 \rrbracket)(u, v); v \vDash_{\Omega(\mathbf{L})} \rho) \right) \right) \\
= & \quad \{ \text{definition of } \llbracket _ \rrbracket \} \\
& \sum_{u \in W} \left(\varphi(u); \left(\sum_{v \in W} (\psi(v); v \vDash_{\Omega(\mathbf{L})} \rho) \right) \right)
\end{aligned}$$

Let us compute the value of variable power in the aggregated state u , as follows:

$$\begin{aligned}
u(\text{power})(8) &= \pi_2(\llbracket (p_1 \wedge p_3)? \rrbracket)(u_1); u_1(\text{power})(8) \\
&\quad + \pi_2(\llbracket (p_1 \wedge p_4)? \rrbracket)(u_2); u_2(\text{power})(8) \\
&\quad + \pi_2(\llbracket (p_2 \wedge p_4)? \rrbracket)(u_3); u_3(\text{power})(8) \\
&= (w_0 \vDash_{\Omega(\mathbf{L})} p_1 \wedge p_3); \top + (w_0 \vDash_{\Omega(\mathbf{L})} p_1 \wedge p_4); \perp \\
&\quad + (w_0 \vDash_{\Omega(\mathbf{L})} p_2 \wedge p_4); \top \\
&= (w_0 \vDash_{\Omega(\mathbf{L})} p_1 \wedge p_3) + (w_0 \vDash_{\Omega(\mathbf{L})} p_2 \wedge p_4) \\
u(\text{power})(0) &= \pi_2(\llbracket (p_1 \wedge p_3)? \rrbracket)(u_1); u_1(\text{power})(0) \\
&\quad + \pi_2(\llbracket (p_1 \wedge p_4)? \rrbracket)(u_2); u_2(\text{power})(0) \\
&\quad + \pi_2(\llbracket (p_2 \wedge p_4)? \rrbracket)(u_3); u_3(\text{power})(0) \\
&= (w_0 \vDash_{\Omega(\mathbf{L})} p_1 \wedge p_3); \perp + (w_0 \vDash_{\Omega(\mathbf{L})} p_1 \wedge p_4); \top \\
&\quad + (w_0 \vDash_{\Omega(\mathbf{L})} p_2 \wedge p_4); \perp \\
&= (w_0 \vDash_{\Omega(\mathbf{L})} p_1 \wedge p_4)
\end{aligned}$$

and \perp otherwise

Hence, φ is defined as

$$\varphi(u) = \begin{cases} \top & \text{if } u(\text{power})(r) = \begin{cases} (w_0 \vDash_{\Omega(\mathbf{L})} p_1 \wedge p_3) + (w_0 \vDash_{\Omega(\mathbf{L})} p_2 \wedge p_4) & \text{if } r = 8 \\ (w_0 \vDash_{\Omega(\mathbf{L})} p_1 \wedge p_4) & \text{if } r = 0 \\ \perp & \text{otherwise} \end{cases} \\ \perp & \text{otherwise} \end{cases}$$

and ψ as

$$\psi(v) = \begin{cases} \top & \text{if } v(\text{power})(r) = \begin{cases} \top & \text{if } r = \frac{8 \times u(\text{power})(8) \oplus 0 \times u(\text{power})(0)}{u(\text{power})(8) \oplus u(\text{power})(0)} \\ \perp & \text{otherwise} \end{cases} \\ \perp & \text{otherwise} \end{cases}$$

where \oplus is the arithmetic sum.

Let us consider the values of `distance` and `angle` as above, and the context rules Γ . By instantiating with the usual three right residuated lattices, we compute the weight of $v(\text{def_power})(r)$, i.e. the application of command `defuzzify` to variable `power`. Since $\cdot = \min$ in the three instances **L**, **G** and **P**, we obtain the same value for r for those lattices:

$$r = \frac{8 \times \max\{\min\{0.9, 0.8\}, \min\{0.1, 0.2\}\} \oplus 0 \times \min\{0.9, 0.2\}}{\max\{\min\{0.9, 0.8\}, \min\{0.1, 0.2\}\} \oplus \min\{0.9, 0.2\}} = \frac{6.4}{1} = 6.4$$

and thus

$$v(\text{power})(r) = \begin{cases} 1 & \text{if } r = 6.4 \\ 0 & \text{otherwise} \end{cases}$$

Let us assume now the additional contextual rule $\text{power} = 6.4 \rightarrow \text{distance} = 8.5$, which means that by setting the motor power to 6.4 kW, the distance of the container to the target position becomes 8.5 yards. Thus, the weight of (131), instantiated with the usual lattices, is given by:

$$\mathbf{L} : v(\text{def_power})(6.4); \llbracket \rho \rrbracket_u = 1; 0.3 = \max\{0, 1 + 0.3 - 1\} = 0.3$$

$$\mathbf{G} : v(\text{def_power})(6.4); \llbracket \rho \rrbracket_u = \min\{1, 0.3\} = 0.3$$

$$\mathbf{P} : v(\text{def_power})(6.4); \llbracket \rho \rrbracket_u = 1 \times 0.3 = 0.3$$

The value 0.3 is the truth degree representing how 'close' the distance becomes after running this controller, by applying `def_power`.

CONCLUSIONS AND OPEN PROBLEMS

This section concludes this PhD thesis. We investigated algebras, semantics and logics for two classes of weighted computation, that we designate by weighted “single-flow” and weighted “multi-flow”. They were expressed by the imperative programming languages \mathcal{F}_1 and \mathcal{F}_2 , respectively.

The approach was presented in two parts, one for type of computation. In a first step, we defined a generic algebra to act as the equational theory of the computational paradigm at hand. Then we introduced a semantics for programs and assertions, where the former is interpreted in terms of the operators of the algebra. The final step was the development of a method, constructed on top of such a semantics, for generating dynamic logics to reason about each class of programs, including a study of the the axiomatisation.

The following two observations sum up the main contributions of the document.

Part 1 focused on the semantic structures, both syntactic and relational, and logics at a propositional level, for weighted “single-flow” computations. We show that the obtained algebras indeed capture weighted programs and assertions, by presenting semantic structures based on fuzzy set theory [Zad65]. The bridge to verification is achieved through an encoding of both PCA’s and deductive system of propositional Hoare logic in these structures, providing a quasi-equational way of reasoning about weighted “single-flow” computations. The research path on the development of multi-valued dynamic logics, initiated in [MNM16], is instantiated by considering programs as (weighted) assignments of terms to variables. The semantics of each logic models variables as weighted sets and programs as weighted functions, taking values over a generic algebra to interpret the notion of weight. Due to the parametric nature of our approach, we were able to instantiate logics to verify programs interpreted as classic assignments, truth degrees and resources consumed in the execution.

Part 2 studied the class of weighted “multi-flow” computations. We introduced a program algebra, with an operator to model weighted parallel execution of programs, of which conditionals in FAS and jFuzzyLogic are examples. The semantics of this kind of computations was given as weighted binary multirelations, with the notion of weight introduced from a complete right residuated lattice. A family of $*$ -free dynamic logics was generated from this semantics, with modal operators adapted from [Pel87] to consider weights in the execution of programs.

We believe this work provides a solid framework which can serve as a basis for the design and verification of programs interpreted in some sort of weighted domain. Although both kinds of single and multi-flow computations live in such a weighted domain, their nature impose that distinct algebras and logics suitable to each of them should be considered, exposing their singularities. Those distinctions are highlighted first, at an abstract level, in the semantics structures and program interpretation for each kind of computation and second, at a more concrete level, in the illustrations presented.

Despite of them, both frameworks resort in two central pillars. One is the proposed relational semantics based on fuzzy set theory. The formalisation, however, follows distinct ways: in **Part 1** programs are modelled as weighted relations, intuitively binary “single flows” of execution; in **Part 2** weighted sets are the second element of an input-output pair of a binary multirelation, representing the (possible multiple) weights of a parallel execution.

The other pillar is the parametric nature of both methods. By adopting a generic algebraic structure to model both computations and assertions, we are able to interpret the weights of both components in a variety of contexts.

Some results in the thesis were confirmed with the help of the automated theorem prover Prover9 [Pro] and its counterexample generator Mace4. Concretely, Prover9 was used to verify the quasi-equational proofs of sections 3.2 and 3.3, and the folk theorem illustration of Section 3.5. Mace4 was utilised to generate the counterexamples discussed in **Part 1**. In such context, we refer to paper [HS07], where the authors axiomatise diverse variants of Kleene algebras, including [MS06], and provide automated proofs of Hoare, dynamic and temporal logics, and concurrency control. Other examples of theorem provers found in the literature for quasi-equational reasoning of programs are KAT-ML [AHK06] and KAT+b! [GKM14], which are specifically designed for reasoning with KAT. The later, in particular, extends KAT with the notion of *mutable test*, allowing to perform certain program transformations without the need to adopt a first-order structure. Contrary to Prover9, which is fully automated, these two provers require some interaction by the user.

There is a set of problems that remain open for future discussion. First of all the problem of defining a notion of bisimulation and proving modal invariance for the logics introduced in **Part 2**, analogously to what was done in **Part 1**, was not conducted and is also clearly in our research agenda. As we already mentioned, continuing the analysis of the three typed of sequential composition for weighted multirelations seems relevant enough to constitute a priority in the near future.

Along the entire document, we gave particular attention to the importance of the parametric nature of our framework, more specifically in the illustrations presented. We believe that such generic approach may benefit further research on fuzzy program analysis. Referring to **Part 2**, it would be possible to extend the range of applications to other non trivial computational domains, such as resource management by considering the tropical lattice of Example 2.2.11.

Note also that some alternatives could be taken regarding the mathematical structure chosen as parameter for the logic of **Part 1**. In particular, note that we choose the \rightarrow operator to be applied only on the subalgebra of tests instead of the whole algebra. The main reason was that we wanted to use such an operator only to model the implication of the logic, and it had no practical purpose, in the context of

this work, to be used on programs as well. An alternative approach could be the use, as a parameter, an action algebra [Koz94b], where the residuation acts on the entire set. Which implications such a modification would have on the thesis, and to study the meaning of defining \rightarrow for programs constitutes a self-contained topic that we left for the future. Another alternative to the chosen structure could be the use of the left residuation operator \leftarrow , instead of \rightarrow . We also believe that such an approach could lead to some implications on the results on this thesis. For instance, property (81) holds from the left residuation, and would not be necessary to impose it to prove the sequential composition of the box operator, as it occurred for Lemma 5.1.5.

Beyond the range of possible research directions specific to **Part 1** or **Part 2**, we enumerate some open problems that are transversal to the entire thesis and we believe that are shared by the two classes of computations addressed. The discussion about those topics is done below.

Discussing Hoare logic in weighted computations

Although Hoare logic is seen as the cornerstone of program correctness, the complex nature of many current systems, namely the ones which serve as the motivation for this thesis, entail the development of novel logics for their rigorous design and verification. Literature provides diverse variants of Hoare logic to reason about programs living in those settings [dHdV02, GMB17, RZ15]. However, in all these approaches, even when some forms of structured computations are taken into consideration, the validity of assertions is always stated in Boolean terms. We believe that it is possible to go a step further and discuss the validity of such structured computation (eg. probabilistic, fuzzy) in a logic capturing itself its inherent behaviour. The statement is supported by the research line already started in this direction [MNM16]. In particular, the family of logics introduced in that reference allows to reason about the validity of *modus ponens*

$$\frac{\rho_1, \rho_1 \rightarrow \rho_2}{\rho_2} \quad (132)$$

and *necessity*

$$\frac{\rho_1}{[\pi]\rho_1} \quad (133)$$

as follows: for each generated multi-valued dynamic logic $\mathcal{GDL}(\mathbf{A})$ (Section 2.3), *modus ponens* is sound iff for any $w \in W$ $(w \models \rho_1) = \top$ and $(w \models \rho_1 \rightarrow \rho_2) = \top$ then $(w \models \rho_2) = \top$; analogously *necessity* is sound iff $(w \models \rho) = \top$ then $(w \models [\pi]\rho) = \top$.

Note that the satisfaction relation of $\mathcal{GDL}(\mathbf{A})$ is defined as a function with codomain in an action lattice \mathbf{A} . The possibility of evaluating a logic formula as a value in a weighted truth space paves the way to go even further and discuss, for example, the soundness of an inference rule in similar terms. Generically, let us consider a signature $\Delta = (\text{Prog}_0, \text{Prop})$, a set of formulæ $\rho_1, \dots, \rho_n, \rho$ in the set of $\mathcal{GDL}(\mathbf{A})$ -formulæ for Δ , $\text{Fm}^{\mathcal{GDL}(\mathbf{A})}(\Delta)$, $n \in \mathbb{N}$ and a rule of inference

$$\frac{\rho_1, \dots, \rho_n}{\rho} \quad (134)$$

Its soundness can be stated in various semantics: classically, as presented in [HKT00], and in a multi-valued setting, as introduced by [MNM16], for a model M in the set of $\mathcal{GDL}(\mathbf{A})$ -models over Δ , $\text{Mod}^{\mathcal{GDL}(\mathbf{A})}(\Delta)$, and a state $w \in W$, as presented in tables 5 and 6.

	Satisfied
S_1	$(\bigwedge_{i \leq n} (M, w \vDash \rho_i)) \Rightarrow M, w \vDash \rho$
S_2	$((\bigwedge_{i \leq n} (M, w \vDash \rho_i) = \top) \Rightarrow ((M, w \vDash \rho) = \top))$
S_3	$\bigwedge_{i \leq n} (M, w \vDash \rho_i) \leq M, w \vDash \rho$

Table 5: Satisfiability

	Valid in M /Globally satisfied
SG_1	$(\bigwedge_{i \leq n} (\forall_{w \in W} (M, w \vDash \rho_i))) \Rightarrow \forall_{w \in W} (M, w \vDash \rho)$
SG_2	$(\bigwedge_{i \leq n} (\bigwedge_{w \in W} (M, w \vDash \rho_i) = \top)) \Rightarrow (\bigwedge_{w \in W} (M, w \vDash \rho) = \top)$
SG_3	$\bigwedge_{i \leq n} (\bigwedge_{w \in W} (M, w \vDash \rho_i)) \leq \bigwedge_{w \in W} (M, w \vDash \rho)$

Table 6: Global satisfiability

Semantics S_1 and SG_1 correspond to the classical case [HKT00], in which the formulæ are evaluated in a Boolean logic. Their difference is explained as follows: a formula ρ is said to be *satisfied in a model* M at w iff $M, w \vDash \rho$; and *globally satisfied* in M , or alternatively, *valid* in M , if $M, w \vDash \rho$ for all states $w \in W$, written $M \vDash \rho$. Semantics S_2 and SG_2 refer to the satisfiability stated in [MNM16]: the logic formulæ are evaluated in an action lattice, but the soundness of a rule is stated in Boolean terms i.e. iff they evaluate to \top (representing true). Finally S_3 and SG_3 use the evaluation of the logic formulæ to express the soundness of the inference rule.

Intuitively, an inference rule is sound in S_3 iff the truth degree of the infimum of the premises is less than or equal than the truth degree of the conclusion. Soundness in SG_3 is stated by considering the infimum of the truth degrees of the formula in all states of a model. In particular, the soundness of *modus ponens* and *necessity* can be studied relatively to the semantics presented in tables 5 and 6.

We are mainly concerned, however, in extending semantics S_3 and SG_3 to other application scenarios, such as Hoare logic. As it is well known [HKT00], the validity of a Hoare triple $\{\rho_1\}\pi\{\rho_2\}$ corresponds, in PDL, to the satisfiability of the formula $\rho_1 \rightarrow [\pi]\rho_2$. Transposing such an encoding to $\mathcal{GDL}(\mathbf{A})$, the value of

$$M, w \vDash \rho_1 \rightarrow [\pi]\rho_2$$

for an action lattice \mathbf{A} , a model $M \in \text{Mod}^{\mathcal{GDL}(\mathbf{A})}(\Delta)$ and a state $w \in W$ gives a truth degree for the same Hoare triple. Intuitively, when the action lattice instantiating the logic is the one of examples 2.2.4, 2.2.6 or 2.2.7, we can interpret such a value as the “degree of correctness” of program π , i.e. the truth degree that the program is correct.

In this semantics, the weights associated to the execution of a program and to assertions about the states of a computation are reflected in the very notion of program correctness, expressed by a truth value in the logic. Thus, it seems natural that the study of (weighted) soundness of *modus ponens* and *necessity* could be extended to the deductive system of Hoare logic. Moreover, due to the parametric nature of our framework, the weight obtained for a Hoare triple may be even given in other, more unconventional contexts besides ‘uncertainty’. For example, by instantiating $\mathcal{GDL}(\mathbf{A})$ with action lattice 2.2.11, instead of reasoning about the correctness of a program, a Hoare triple $\{\rho_1\}\pi\{\rho_2\}$ may represent the energy consumed by running π , limited by some sort of energy restrictions before (ρ_1) and after (ρ_2) its execution. How to interpret those values in a real scenario of program verification, and how such an interpretation can be used to expand the research about program correctness to unconventional contexts seems a challenging research topic.

A predicate transformer semantics for weighted programs?

Following a similar approach, a future research path we may pursue is the discussion of *wlp* semantics in a weighted domain, due to its strict relation with partial correctness in Hoare logic. The general idea is to define, for an action lattice \mathbf{A} , the *weighted weakest liberal precondition semantics (wwlp)* as a function

$$wwlp : \text{Prog} \times \text{Fm}^{\mathcal{GDL}(\mathbf{A})} \rightarrow \mathbf{A}^W$$

defined as

$$wwlp(\pi, \rho)(w) = w \models [\pi]\rho$$

The weight of $w \models [\pi]\rho$ will be obtained, of course, using the satisfaction relation of $\mathcal{GDL}(\mathbf{A})$, given in Chapter 2. An adaptation can be naturally made to include variables and assignments. The classic *wlp* semantics is bundled with a set of properties which make it possible to construct valid deductions of Hoare logic, for standard imperative programming constructs. Analogously to the Boolean case, the discussion, in the *wwlp* semantics, of the validity of these properties, their interpretation, and relation with partial correctness in Hoare logic is worth to be studied. A similar approach for generalising predicate transformer semantics was taken in [CW08]. However, the truth space in which the weights are evaluated there is restricted to the continuous interval $[0, 1]$, while our approach intends to be, like the path pursued in this thesis, parametric on a generic mathematical structure, such as an action lattice.

Another related study on predicate transformer semantics in this direction was taken by [MMS96, MM01b], although oriented to probabilistic programs. The authors generalise pre and post conditions to probabilistic predicates, i.e. real-valued expressions over a state space. Thus reasoning about the cor-

rectness of a probabilistic program is given in terms of those (probabilistic) conditions. For example, for a probabilistic program [MMS96]

$$(n := 1) \oplus_{\frac{1}{2}} (n := 2) \tag{135}$$

the weakest-precondition (*wp*) formulation yields

$$wp.(n := 1 \oplus_{\frac{1}{2}} n := 2).(n = 1) = \frac{1}{2}$$

where $\frac{1}{2}$ is the probability that program (135) establishes $n = 1$. The weighted nature of such a formalisation, and the corresponding intuitive interpretation constitute naturally also a reference for the approach that we want to pursue, even if we target, obviously, a class of programs of a different nature. In our framework, the value $\frac{1}{2}$ could be, for instance, the truth degree with which a program reaches a given state, or the cost (energy, time) of such an execution.

To sum up, we are interested in discussing generalisations of Hoare logic and predicate transformer semantics for \mathcal{F}_1 and \mathcal{F}_2 programs, and study the implications of their weighted behaviour in these formalisations and related properties.

A table revisited

We conclude this thesis with the table below, presenting a wrap up of the main formalisms introduced along the document. We add to Table 2 some algebras and logics we want to explore in future work, marked with brown-coloured X marks.

As previously discussed, a Hoare logic and a predicate transformer semantics is missing for the majority of the frameworks in the thesis. Following an analogous approach to Section 5.4, we want also to contribute with a notion of bisimulation and study its modal invariance for $\Omega(\mathbf{L})$. Additionally, we are interested to develop equational and weighted versions of classic CPDL. The former could be compared with proof calculi for concurrent programs [HMSW11], while the latter may represent an abstraction of structure $\Omega(\mathbf{L})$ introduced in this thesis. Note that, although not presented in this thesis, and thus marked with brown-coloured X marks in the table, a weighted version of CPDL [Pel87] is documented in [Gom20].

	Prop	Eq	Rel	MRel	B	W
Hoare Logic (HL) [Hoa69]		X	X		X	
Propositional HL (PHL) [Koz00]	X		X		X	
Predicate transformer (wlp) [Dij76]	X	X	X		X	
Propositional Dynamic Logic (PDL) [FL77]	X		X		X	
First-order DL [HKT00]		X	X		X	
Kleene Algebra with Tests (KAT) [Koz97]	X		X		X	
Kleene Algebra with Domain (KAD) [DMS06]	X		X		X	
Probabilistic HL [dHdV02]		X	X			X
Probabilistic wlp [MMS96]		X	X			X
Probabilistic PDL [Koz85]	X		X			X
Probabilistic KAT [QWWG08]	X		X			X
Weighted PDL ($\mathcal{GDL}(\mathbf{A})$) [MNM16]	X		X			X
GKAT /I-GKAT [GMB19]	X		X			X
Weighted PHL [GMB17]	X		X			X
Weighted Equational Dynamic Logic ($\Gamma(\mathbf{A})$) [GMJB19]		X	X			X
Weighted HL		X	X			X
Weighted wlp		X	X			X
“Weighted” KAD [DS11]	X		X			X
Algebra of binary multirelations [Rew03]	X			X	X	
Concurrent PDL (CPDL) [Pel87]	X			X	X	
Equational “Multi-flow” DL		X		X	X	
“Multi-flow” HL	X	X		X	X	
“Multi-flow” wlp		X		X	X	
Concurrent KAT (CKAT) [JM16]	X		X		X	
Concurrent Dynamic Algebra [FS15]	X			X	X	
Probabilistic CKA [MRS13]	X		X			X
Algebra of probabilistic multirelations [Tsu12]	X			X		X
Algebra of weighted binary multirelations	X			X		X
Weighted CPDL [Gom20]	X			X		X
Weighted “multi-flow” dynamic logic ($\Omega(\mathbf{L})$)		X		X		X
Weighted “Multi-flow” HL	X	X		X		X
Weighted “Multi-flow” wlp		X		X		X

Table 7: Taxonomy of related work, frameworks introduced in this thesis and additional ones we left for the future

BIBLIOGRAPHY

- [ACB⁺18] Vibha Anand, Aaron E. Carroll, Paul G. Biondich, Tamara M. Dugan, and Stephen M. Downs. Pediatric decision support using adapted arden syntax. *Artificial Intelligence in Medicine*, 92:15–23, 2018.
- [AFG⁺14] Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. Netkat: semantic foundations for networks. In Suresh Jagannathan and Peter Sewell, editors, *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, 2014*, POPL '14, pages 113–126. Association for Computing Machinery, 2014.
- [AFKW14] Carolyn Jane Anderson, Nate Foster, Dexter Kozen, and David Walker. NetKAT : Semantic Foundations for Networks. In Suresh Jagannathan and Peter Sewell, editors, *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14*, pages 113–126. Association for Computing Machinery, 2014.
- [AHK06] Kamal Aboul-Hosn and Dexter Kozen. KAT-ML: An interactive theorem prover for Kleene algebra with tests. *Journal of Applied Non-Classical Logics*, 16(1-2):9–33, 2006.
- [AZ96] Lofti A. Zadeh. *Fuzzy languages and their relation to human and machine intelligence*, pages 148–179. World Scientific Publishing Co., Inc., USA, 1996.
- [BEGR11] Félix Bou, Francesc Esteva, Lluís Godo, and Ricardo Oscar Rodríguez. On the minimum many-valued modal logic over a finite residuated lattice. *Journal of Logic and computation*, 21(5):739–790, 2011.
- [BF00] W. J. Blok and I. Ferreirim. On the structure of hoops. *Algebra Universalis*, 43(2-3):233–257, 2000.
- [BKW16] Bernhard Beckert, Vladimir Klebanov, and Benjamin Weiß. Dynamic logic for java. In Wolfgang Ahrendt, Bernhard Beckert, Richard Bubel, Reiner Hähnle, Peter H. Schmitt, and Matthias Ulbrich, editors, *Deductive Software Verification - The KeY Book*, volume 10001 of *Lecture Notes in Computer Science*, pages 49–106. Springer, Cham, 2016.
- [BM89] Bard Bloom and Albert R. Meyer. A remark on bisimulation between probabilistic processes. In Albert R. Meyer and Michael A. Taitslin, editors, *Logic at Botik '89, Symposium on Logical Foundations of Computer Science, Pereslav-Zalessky, USSR, 1989, Proceedings*, volume 363, pages 26–40. Springer, Berlin, Heidelberg, 1989.

- [BS06] Alexandru Baltag and Sonja Smets. LQP: the dynamic logic of quantum information. *Mathematical Structures in Computer Science*, 16(3):491–525, 2006.
- [BS09] Walter Binder and Niranjan Suri. Green computing: Energy consumption optimized service hosting. In Mogens Nielsen, Antonín Kučera, Peter Bro Miltersen, Catuscia Palamidessi, Petr Tůma, and Frank Valencia, editors, *SOFSEM 2009: Theory and Practice of Computer Science*, pages 117–128. Springer Berlin Heidelberg, 2009.
- [BS12] Alexandru Baltag and Sonja Smets. The dynamic turn in quantum logic. *Synthese*, 186(3):753–773, 2012.
- [BvBW06] Patrick Blackburn, J. F. A. K. van Benthem, and Frank Wolter, editors. *Handbook of Modal Logic*, volume 3 of *Studies in logic and practical reasoning*. Elsevier Science, 2006.
- [CA12] Pablo Cingolani and Jesús Alcalá-Fdez. jfuzzylogic: a robust and flexible fuzzy-logic inference system language implementation. In *FUZZ-IEEE 2012, IEEE International Conference on Fuzzy Systems, Brisbane, Australia, 2012, Proceedings*, pages 1–8. IEEE, 2012.
- [CAf13] Pablo Cingolani and Jesús Alcalá-fdez. jFuzzyLogic: a Java library to design fuzzy logic controllers according to the standard for fuzzy control programming. *International Journal of Computational Intelligence Systems*, 6:61–75, 2013.
- [CL14] Rajani Chulyadyo and Philippe Leray. A personalized recommender system from probabilistic relational model and users’ preferences. In Piotr Jędrzejowicz, Lakhmi C. Jain, Robert J. Howlett, and Ireneusz Czarnowski, editors, *18th International Conference in Knowledge Based and Intelligent Information and Engineering Systems, KES 2014, Gdynia, Poland, 2014*, volume 35 of *Procedia Computer Science*, pages 1063–1072. Elsevier, 2014.
- [Con12] J.H. Conway. *Regular Algebra and Finite Machines*. Dover Books on Mathematics. Dover Publications, 2012.
- [CSLM19] Marco F. Cusumano-Towner, Feras A. Saad, Alexander K. Lew, and Vikash K. Mansinghka. Gen: a general-purpose probabilistic programming system with programmable inference. In Kathryn S. McKinley and Kathleen Fisher, editors, *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019, Phoenix, AZ, USA, PLDI 2019*, pages 221–236, New York, NY, USA, 2019. Association for Computing Machinery.
- [CW08] Yixiang Chen and Hengyang Wu. Semantics of sub-probabilistic programs. *Frontiers of Computer Science in China*, 2(1):29–38, 2008.
- [dHdV02] Jerry den Hartog and Erik P. de Vink. Verifying probabilistic programs using a Hoare like logic. *International journal of foundations of computer science*, 13(3):315–340, 2002.

- [DHM11] H.-H. Dang, P. Höfner, and B. Möller. Algebraic separation logic. *The Journal of Logic and Algebraic Programming*, 80(6):221 – 247, 2011.
- [Dij76] Edsger W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.
- [Dij98] Rutger M. Dijkstra. Computation calculus bridging a formalization gap. In Johan Jeuring, editor, *Mathematics of Program Construction*, volume 1422 of *Lecture Notes in Computer Science*, pages 151–174, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.
- [Dij00] Rutger M. Dijkstra. Computation calculus bridging a formalization gap. *Science of Computer Programming*, 37(1):3 – 36, 2000.
- [DM14] Jules Desharnais and Bernhard Möller. Fuzzifying modal algebra. In Peter Höfner, Peter Jipsen, Wolfram Kahl, and Martin Eric Müller, editors, *Relational and Algebraic Methods in Computer Science - 14th International Conference, RAMiCS 2014, Marienstatt, Germany, 2014. Proceedings*, volume 8428 of *Lecture Notes in Computer Science*, pages 395–411. Springer, 2014.
- [DMS06] J. Desharnais, B. Möller, and G. Struth. Kleene algebra with domain. *ACM Transactions on Computational Logic (TOCL)*, 7(4):798–833, 2006.
- [DMS11] J. Desharnais, B. Möller, and G. Struth. Algebraic notions of termination. *Logical Methods in Computer Science*, 7(1), 2011.
- [DS08] Jules Desharnais and Georg Struth. Modal semirings revisited. In Philippe Audebaud and Christine Paulin-Mohring, editors, *Mathematics of Program Construction, 9th International Conference, MPC 2008, Marseille, France, 2008. Proceedings*, volume 5133 of *Lecture Notes in Computer Science*, pages 360–387. Springer, 2008.
- [DS11] Jules Desharnais and Georg Struth. Internal axioms for domain semirings. *Science of Computer Programming*, 76(3):181–203, 2011.
- [EC96] F. Smith E. Cohen, D. Kozen. The complexity of Kleene algebra with tests. Technical report, Computer Science Department, Cornell University, Ithaca, NY 14853-7501, USA, 1996.
- [FD07] Therrezinha Fernandes and Jules Desharnais. Describing data flow analysis techniques with kleene algebra. *Science of Computer Programming*, 65(2):173 – 194, 2007.
- [FH84] Yishai A. Feldman and David Harel. A probabilistic dynamic logic. *Journal of Computer and System Sciences*, 28(2):193–215, 1984.
- [Fit91] Melvin C. Fitting. Many-valued modal logics. *Fundamenta Informaticae*, 15(3-4):235–254, 1991.
- [Fit92a] Melvin Fitting. Many-Valued Modal Logics II. *Fundamenta Informaticae*, 17(1-2):55–73, 1992.

- [Fit92b] Melvin Fitting. Many-Valued Modal Logics II. *Fundamenta Informaticae*, 17(1-2):55–73, 1992.
- [FKM⁺16] Nate Foster, Dexter Kozen, Konstantinos Mamouras, Mark Reitblatt, and Alexandra Silva. Probabilistic netkat. In Peter Thiemann, editor, *Programming Languages and Systems - 25th European Symposium on Programming, ESOP 2016, Held as Part of ETAPS 2016, Eindhoven, The Netherlands, Proceedings*, volume 9632 of *Lecture Notes in Computer Science*, pages 282–309. Springer Berlin Heidelberg, 2016.
- [FKST17] Hitoshi Furusawa, Yasuo Kawahara, Georg Struth, and Norihiro Tsumagari. Kleisli, Parikh and Peleg compositions and liftings for multirelations. *Journal of Logical and Algebraic Methods in Programming*, 90:84–101, 2017.
- [FL77] Michael J. Fischer and Richard E. Ladner. Propositional modal logic of programs. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing, Boulder, Colorado, USA 1977*, STOC '77, page 286–294, New York, NY, USA, 1977. Association for Computing Machinery.
- [FL79] Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194–211, 1979.
- [Flo93] R. W. Floyd. Assigning meanings to programs. In T. R. Colburn, J. H. Fetzer, and T. L. Rankin, editors, *Program Verification*, volume 14 of *Studies in Cognitive Systems*, pages 65–81. Springer Netherlands, Dordrecht, 1993.
- [FS15] Hitoshi Furusawa and Georg Struth. Concurrent dynamic algebra. *ACM Transactions on Computational Logic (TOCL)*, 16(4):1–38, 2015.
- [FS16] Hitoshi Furusawa and Georg Struth. Taming multirelations. *ACM Transactions on Computational Logic (TOCL)*, 17(4):1–34, 2016.
- [GJK007] Nikolaos Galatos, Peter Jipsen, Tomasz Kowalski, and Hiroakira Ono. *Residuated lattices: an algebraic glimpse at substructural logics*, volume 151. Elsevier, 1 edition, 2007.
- [GKM14] Niels Grathwohl, Dexter Kozen, and Konstantinos Mamouras. Kat + b! In *Proceedings of the Joint Meeting of the 23rd EACSL Annual Conference on CSL and the 29th Annual LICS, CSL-LICS '14*, New York, NY, USA, 2014. Association for Computing Machinery.
- [GMB17] Leandro Gomes, Alexandre Madeira, and Luís Soares Barbosa. On Kleene algebras for weighted computation. In Simone André da Costa Cavalheiro and José Luiz Fiadeiro, editors, *Formal Methods: Foundations and Applications, Brazilian Symposium on Formal Methods SBMF 2017, Recife, Brazil, Proceedings*, volume 10623 of *Lecture Notes in Computer Science*, pages 271–286. Springer, Cham, 2017.
- [GMB19] Leandro Gomes, Alexandre Madeira, and Luís Soares Barbosa. Generalising KAT to verify weighted computations. *Scientific Annals of Computer Science*, 29(2):141–184, 2019.

- [GMJB19] Leandro Gomes, Alexandre Madeira, Manisha Jain, and Luís Soares Barbosa. On the generation of equational dynamic logics for weighted imperative programs. In Yamine Ait Ameur and Shengchao Qin, editors, *Formal Methods and Software Engineering - International Conference on Formal Engineering Methods, ICFEM 2019, Shenzhen, China, 2019, Proceedings*, volume 11852 of *Lecture Notes in Computer Science*, pages 154–169. Springer, 2019.
- [Gog67] J.A Goguen. L-fuzzy sets. *Journal of Mathematical Analysis and Applications*, 18(1):145 – 174, 1967.
- [Gom20] Leandro Gomes. On the construction of multi-valued concurrent dynamic logic. In Baltag A. Soares Barbosa L., editor, *Dynamic Logic. New Trends and Applications. DALI 2019*, volume 12005 of *Lecture Notes in Computer Science*, pages 218–226. Springer, Cham, 2020.
- [HCHH19] De-Thu Huynh, Min Chen, Trong-Thua Huynh, and Chu Hong Hai. Energy consumption optimization for green device-to-device multimedia communications. *Future Generation Computer Systems*, 92:1131–1141, 2019.
- [HKT00] David Harel, Dexter Kozen, and Jerzy Tiuryn. *Dynamic logic*. MIT Press, Cambridge, MA, USA, 2000.
- [HM09] P. Höfner and B. Möller. An algebra of hybrid systems. *Journal of Logical and Algebraic Methods in Programming*, 78(2):74–97, 2009.
- [HMK19] R. Hennicker, A. Madeira, and A. Knapp. A hybrid dynamic logic for event/data-based systems. In Reiner Hähnle and Wil M. P. van der Aalst, editors, *Fundamental Approaches to Software Engineering - FASE 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, 2019, Proceedings*, volume 11424 of *Lecture Notes in Computer Science*, pages 79–97. Springer, Cham, 2019.
- [HMSW11] Tony Hoare, Bernhard Möller, Georg Struth, and Ian Wehrman. Concurrent kleene algebra and its foundations. *Journal of Logic and Algebraic Programming*, 80(6):266–296, 2011.
- [Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.
- [Hri94] George Hripcsak. Writing arden syntax medical logic modules. *Computers in Biology and Medicine*, 24(5):331 – 363, 1994.
- [HS07] Peter Höfner and Georg Struth. Automated reasoning in kleene algebra. In Pfenning F., editor, *International Conference on Automated Deduction - CADE 21*, volume 4603 of *Lecture Notes in Computer Science*, pages 279–294. Springer, Berlin, Heidelberg, 2007.
- [HSM97] Jifeng He, Karen Seidel, and Annabelle McIver. Probabilistic models for the guarded command language. *Science of Computer Programming*, 28(2-3):171–192, 1997.

- [HWL⁺08] James S. Hutchison, Roxanne E. Ward, Jacques Lacroix, Paul C. Hébert, Marcia A. Barnes, Desmond J. Bohn, Peter B. Dirks, Steve Doucette, Dean Fergusson, Ronald Gottesman, Ari R. Joffe, Haresh M. Kirpalani, Philippe G. Meyer, Kevin P. Morris, David Moher, Ram N. Singh, and Peter W. Skippen. Hypothermia therapy after traumatic brain injury in children. *New England Journal of Medicine*, 358(23):2447–2456, 2008.
- [jfu] jfuzzylogic. http://jfuzzylogic.sourceforge.net/doc/iec_1131_7_cd1.pdf.
- [JM16] Peter Jipsen and M. Andrew Moshier. Concurrent Kleene algebra with tests and branching automata. *Journal of Logical and Algebraic Methods in Programming*, 85(4):637–652, 2016.
- [JMM20] M. Jain, A. Madeira, and M. A. Martins. A fuzzy modal logic for fuzzy transition systems. *Electronic Notes in Theoretical Computer Science*, 348:85–103, 2020.
- [KAK⁺96] Yuhei Kawano, Hitoshi Abe, Shunichi Kojima, Hiroki Yoshimi, Toru Sanai, Genjiro Kimura, Hiroaki Matsuoka, Shuichi Takishita, and Teruo Omae. Different effects of alcohol and salt on 24-hour blood pressure and heart rate in hypertensive patients. *Hypertension Research*, 19(4):255–261, 1996.
- [Kle56] Stephen C. Kleene. *Representation of events in nerve nets and finite automata*, volume 34, pages 3–42. Princeton University Press, Princeton, NJ, 1956.
- [KMRG15] Alexander Knapp, Till Mossakowski, Markus Roggenbach, and Martin Glauer. An institution for simple UML state machines. In Alexander Egyed and Ina Schaefer, editors, *Fundamental Approaches to Software Engineering (FASE)*, volume abs/1411.4495, pages 3–18, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.
- [Koz81] Dexter Kozen. Semantics of probabilistic programs. *Journal of Computer and System Sciences*, 22(3):328–350, 1981.
- [Koz82] Dexter Kozen. On induction vs. \ast -continuity. In Dexter Kozen, editor, *Logics of Programs*, Lecture Notes in Computer Science, pages 167–176. Springer, Berlin, Heidelberg, 1982.
- [Koz85] Dexter Kozen. A probabilistic PDL. *Journal of Computer and System Sciences*, 30(2):162–178, 1985.
- [Koz92] Dexter Kozen. *The design and analysis of algorithms*. Texts and Monographs in Computer Science. Springer-Verlag New York, 1992.
- [Koz94a] Dexter Kozen. A Completeness Theorem for Kleene Algebras and the Algebra of Regular Events. *Information and Computation*, 110(2):366–390, 1994.
- [Koz94b] Dexter Kozen. *On Action Algebras*, pages 78–88. MIT Press, 1994.

- [Koz97] Dexter Kozen. Kleene algebra with tests. *ACM Transactions on Programming Languages and Systems*, 19(3):427–443, 1997.
- [Koz00] Dexter Kozen. On hoare logic and kleene algebra with tests. *ACM Transactions on Computational Logic (TOCL)*, 1(1):60–76, 2000.
- [Koz14] Dexter Kozen. Netkat—a formal system for the verification of networks. In Garrigue J., editor, *Programming Languages and Systems (APLAS 2014)*, volume 8858 of *Lecture Notes in Computer Science*, pages 1–18. Springer, Cham, 2014.
- [KTMK15] Tejas Kulkarni, Joshua Tenenbaum, Vikash Mansinghka, and Pushmeet Kohli. Picture: A probabilistic programming language for scene perception, 2015.
- [Kur08] Patrick Kurp. Green computing. *Communications of the ACM*, 51(10):11–13, 2008.
- [LS91] Kim Guldstrand Larsen and Arne Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.
- [Luk93] Johan J. Lukkien. An operational semantics for the guarded command language. In R. S. Bird, C. C. Morgan, and J. C. P. Woodcock, editors, *Mathematics of Program Construction*, volume 669 of *Lecture Notes in Computer Science*, pages 233–249, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.
- [Luk94] Johan J. Lukkien. Operational semantics and generalized weakest preconditions. *Science of Computer Programming*, 22(1):137 – 155, 1994.
- [LY02] Feng Lin and Hao Ying. Modeling and control of fuzzy discrete event systems. *IEEE Trans. Systems, Man, and Cybernetics, Part B*, 32(4):408–415, 2002.
- [Mac71] Saunders MacLane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer-Verlag New York, 1971.
- [MCM06] Annabelle McIver, E. Cohen, and Carroll Morgan. Using probabilistic kleene algebra for protocol verification. In Renate A. Schmidt, editor, *Relations and Kleene Algebra in Computer Science, 9th International Conference on Relational Methods in Computer Science and 4th International Workshop on Applications of Kleene Algebra, ReIMiCS/AKA 2006, Manchester, UK, Proceedings*, Lecture Notes in Computer Science, pages 296–310. Springer, Berlin, Heidelberg, 2006.
- [MCR07] Clare E. Martin, Sharon A. Curtis, and Ingrid Rewitzky. Modelling angelic and demonic non-determinism with multirelations. *Science of Computer Programming*, 65(2):140–158, 2007.
- [MM01a] Annabelle McIver and Carroll Morgan. Demonic, angelic and unbounded probabilistic choices in sequential programs. *Acta Informatica*, 37(4-5):329–354, 2001.

- [MM01b] Annabelle McIver and Carroll Morgan. Partial correctness for probabilistic demonic programs. *Theoretical Computer Science*, 266(1-2):513–541, 2001.
- [MM05] Annabelle McIver and Carroll Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Monographs in Computer Science. Springer, 2005.
- [MMS96] Carroll Morgan, Annabelle McIver, and Karen Seidel. Probabilistic predicate transformers. *ACM Transactions on Programming Languages and Systems*, 18(3):325–353, 1996.
- [MNM16] Alexandre Madeira, Renato Neves, and Manuel A. Martins. An exercise on the generation of many-valued dynamic logics. *Journal of Logical and Algebraic Methods in Programming*, 85(5):1011–1037, 2016.
- [MNMB15] Alexandre Madeira, Renato Neves, Manuel A. Martins, and Luís Soares Barbosa. A dynamic logic for every season. In Christiano Braga and Narciso Martí-Oliet, editors, *Formal Methods: Foundations and Applications - 17th Brazilian Symposium, SBMF 2014, Maceió, AL, Brazil, Proceedings*, volume 8941 of *Lecture Notes in Computer Science*, pages 130–145. Springer, Cham, 2015.
- [MRS13] Annabelle McIver, Tahiry M. Rabehaja, and Georg Struth. Probabilistic concurrent Kleene algebra. In Luca Bortolussi and Herbert Wiklicky, editors, *Proceedings 11th International Workshop on Quantitative Aspects of Programming Languages and Systems, QAPL 2013, Rome, Italy*, volume 117 of *EPTCS*, pages 97–115, 2013.
- [MS04] Bernhard Möller and Georg Struth. Modal kleene algebra and partial correctness. In Charles Rattray, Savi Maharaj, and Carron Shankland, editors, *Algebraic Methodology and Software Technology, 10th International Conference, AMAST 2004, Stirling, Scotland, UK, 2004, Proceedings*, volume 3116 of *Lecture Notes in Computer Science*, pages 379–393. Springer, 2004.
- [MS06] Bernhard Möller and Georg Struth. Algebras of modal operators and partial correctness. *Theoretical Computer Science*, 351(2):221–239, 2006.
- [MW81] D.S. Minors and J.M. Waterhouse. chapter 2 - the circadian rhythm of deep body temperature. In D.S. Minors and J.M. Waterhouse, editors, *Circadian Rhythms and the Human*, pages 24–40. Butterworth-Heinemann, 1981.
- [Mö07] Bernhard Möller. Kleene getting lazy. *Science of Computer Programming*, 65(2):195 – 214, 2007. Special Issue dedicated to selected papers from the conference of program construction 2004 (MPC 2004).
- [NW05] Hung T. Nguyen and Elbert A. Walker. *A first course in fuzzy logic (3. ed.)*. Chapman&Hall/CRC Press, 2005.

- [Ost82] Walenty Ostasiewicz. A new approach to fuzzy programming. *Fuzzy Sets and Systems*, 7(2):139 – 152, 1982.
- [Par83] Rohit Parikh. Propositional game logic. In *24th Annual Symposium on Foundations of Computer Science, Tucson, Arizona, USA, 1983*, SFCS '83, pages 195–200. IEEE Computer Society, 1983.
- [Pel87] David Peleg. Concurrent dynamic logic. *Journal of the ACM*, 34(2):450–479, 1987.
- [Pla10] André Platzer. *Logical Analysis of Hybrid Systems - Proving Theorems for Complex Dynamics*. Springer, 2010.
- [Pra76] Vaughan R. Pratt. Semantical considerations on floyd-hoare logic. In *17th Annual Symposium on Foundations of Computer Science, Houston, Texas, USA, 25-27 October 1976*, pages 109–121. IEEE Computer Society, 1976.
- [Pra88] Vaughan R. Pratt. Dynamic algebras as a well-behaved fragment of relation algebras. In Clifford Bergman, Roger D. Maddux, and Don Pigozzi, editors, *Algebraic Logic and Universal Algebra in Computer Science, Conference, Ames, Iowa, USA, 1988, Proceedings*, volume 425 of *Lecture Notes in Computer Science*, pages 77–110. Springer, 1988.
- [Pra91] Vaughan R. Pratt. Action logic and pure induction. In Jan van Eijck, editor, *Logics in AI, JELIA 1990 Proceedings*, volume 478 of *Lecture Notes in Computer Science (Lecture Notes in Artificial Intelligence)*, pages 97–120. Springer, Berlin, Heidelberg, 1991.
- [Pri10] Cristian Prisacariu. Synchronous Kleene algebra. *The Journal of Logic and Algebraic Programming*, 79(7):608–635, 2010.
- [Pro] Prover9 and Mace4. <https://www.cs.unm.edu/~mccune/mace4/manual/2009-11A>.
- [QWG08] Rui Qiao, Jinzhao Wu, and Xinyan Gao. Probabilistic modal kleene algebra and hoare-style logic. In Maozu Guo, Liang Zhao, and Lipo Wang, editors, *Fourth International Conference on Natural Computation, ICNC 2008, Jinan, Shandong, China, 2008, Volume 3*, volume 3, pages 652–661. IEEE Computer Society, 2008.
- [QWWG08] Rui Qiao, Jinzhao Wu, Yuan Wang, and Xinyan Gao. Operational semantics of probabilistic Kleene algebra with tests. In *Proceedings - IEEE Symposium on Computers and Communications (ISCC)*, pages 706–713, Marrakech, Morocco, 2008. IEEE Computer Society.
- [Rew03] Ingrid Rewitzky. Binary multirelations. In Harrie C. M. de Swart, Ewa Orłowska, Gunther Schmidt, and Marc Roubens, editors, *Theory and Applications of Relational Structures as Knowledge Instruments, COST Action 274, TARSKI, Revised Papers*, volume 2929 of *Lecture Notes in Computer Science*, pages 256–271. Springer, Berlin, Heidelberg, 2003.

- [RZ15] Robert Rand and Steve Zdancewic. VPHL: A verified partial-correctness logic for probabilistic programs. *Electronic Notes in Theoretical Computer Science*, 319:351–367, 2015.
- [SdV04] Ana Sokolova and Erik P. de Vink. Probabilistic automata: System types, parallel composition and comparison. In Christel Baier, Boudewijn R. Haverkort, Holger Hermanns, Joost-Pieter Katoen, and Markus Siegle, editors, *Validation of Stochastic Systems*, volume 2925 of *Lecture Notes in Computer Science*, pages 1–43. Springer, Berlin, Heidelberg, 2004.
- [Sed20] Igor Sedlár. Finitely-valued propositional dynamic logic. In Nicola Olivetti, Rineke Verbrugge, Sara Negri, and Gabriel Sandu, editors, *13th Conference on Advances in Modal Logic, AiML 2020, Helsinki, Finland, August 24-28, 2020*, volume 13, pages 561–579. College Publications, 2020.
- [SFdBA12] Matthias Samwald, Karsten Fehre, Jeroen de Bruin, and Klaus-Peter Adlassnig. The arden syntax standard for clinical decision support: Experiences and directions. *Journal of Biomedical Informatics*, 45(4):711 – 718, 2012. Translating Standards into Practice: Experiences and Lessons Learned in Biomedicine and Health Care.
- [SHJ⁺94] J.B. Starren, G. Hripcsak, D. Jordan, B. Allen, C. Weissman, and P.D. Clayton. Encoding a post-operative coronary artery bypass surgery care plan in the arden syntax. *Computers in Biology and Medicine*, 24(5):411 – 417, 1994. The Arden Syntax.
- [Tsu12] Norihiro Tsumagari. Probabilistic relational models of complete il-semirings. *Bulletin of Informatics and Cybernetics*, 44:87–109, 2012.
- [VMA10] Thomas Vetterlein, Harald Mandl, and Klaus-Peter Adlassnig. Fuzzy arden syntax: A fuzzy programming language for medicine. *Artificial Intelligence in Medicine*, 49(1):1 – 10, 2010.
- [WLCW20] Yupeng Wang, Tianlong Liu, Chang Choi, and Haoxiang Wang. Green resource allocation method for intelligent medical treatment-oriented service in a 5g mobile network. *Concurrency and Computation: Practice and Experience*, 32(1):e5057, 2020.
- [WLF⁺20] Dominic P. Williams, Stanley E. Lazic, Alison J Foster, Elizaveta A. Semenova, and P. Morgan. Predicting drug-induced liver injury with bayesian machine learning. *Chemical research in toxicology*, 33(1):239–248, 2020.
- [Zad65] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338 – 353, 1965.