# Towards weak bisimilarity on a class of parallel processes.

Harwood, W. J. T

# Towards weak bisimilarity on a class of parallel processes

W. J. T. Harwood BSc. (Wales) MSc. (Oxon)

Submitted to the University of Wales in fulfilment of the requirements for the Degree of Doctor of Philosophy

Swansea University

December 2006

## Declaration

This work has not been previously accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed ............ (candidate)

Date 14/06/2007

## Statement 1

This thesis is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

Signed ...... ...... (candidate)

Date 14/06/2007

## Statement 2

I hereby give my consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed .... ........ (candidate)

Date 14/06/2007

## Abstract

A directed labelled graph may be used, at a certain abstraction, to represent a system's behaviour. Its nodes, the possible states the system can be in; its arrows labelled by the actions required to move from one state to another. *Processes* are, for our purposes, synonymous with these *labelled transition systems*.

With this view a well-studied notion of behavioural equivalence is *bisimilarity*, where processes are *bisimilar* when whatever one can do, the other can match, while maintaining bisimilarity. *Weak bisimilarity* accommodates a notion of silent or internal action. A natural class of labelled transition systems is given by considering the derivations of commutative context-free grammars in Greibach Normal Form: the Basic Parallel Processes (BPP), introduced by Christensen in his PhD thesis. They represent a simple model of communication-free parallel computation, and for them bisimilarity is PSPACE-complete. Weak bisimilarity is believed to be decidable, but only partial results exist.

Non-bisimilarity is trivially semidecidable on BPP (each process has finitely many next states, so the state space can be explored until a mis-match is found); the research effort in proving it fully decidable centred on semideciding the positive case. Conversely, weak bisimilarity has been known to be semidecidable for a decade, but no method for semideciding inequivalence has yet been found – the presence of silent actions allows a process to have infinitely many possible successor states, so simple exploration is no longer possible.

Weak bisimilarity is defined coinductively, but may be approached, and even reached, by its inductively defined *approximants*. Game theoretically, these change the Defender's winning condition from survival for infinitely many turns to survival for $\kappa$ turns, for an ordinal $\kappa$, creating a hierarchy of relations successively closer to full weak bisimilarity. It can be seen that on any *set* of processes this approximant hierarchy collapses: there will always exist some $\kappa$ such that the $\kappa$th approximant coincides with weak bisimilarity. One avenue towards the semidecidability of non-weak bisimilarity is the decidability of its approximants.

It is a long-standing conjecture that on BPP the weak approximant hierarchy collapses at $\omega \times 2$. If true, in order to semidecide inequivalence it would suffice to be able to decide the $\omega + n$ approximants. Again, there exist only limited results: the *finite* approximants are known to be decidable, but no progress has been made on the $\omega$th approximant, and thus far the best proven lower-bound of collapse is $\omega_{1\text{CK}}$ (the least non-recursive ordinal number). We significantly improve this bound

to $\omega^{k \times 2}$ (for a $k$-variable BPP); a key part of the proof being a novel constructive version of Dickson's Lemma.

The distances-to-disablings or DD functions were invented by Jančar in order to prove the PSPACE-completeness of bisimilarity on BPP. At the end of his paper is a conjecture that weak bisimilarity might be amenable to the theory; a suggestion we have taken up.

We generalise and extend the DD functions, widening the subset of BPP on which weak bisimilarity is known to be computable, and creating a new means for testing inequivalence. The thesis ends with two conjectures. The first, that our extended DD functions in fact capture weak bisimilarity on full BPP (a corollary of which would be to take the lower bound of approximant collapse to $\omega^2$); and second, that they are computable, which would enable us to semidecide inequivalence, and hence give us the decidability of weak bisimilarity.

# Acknowledgments

I wish to thank my supervisor, Faron Moller, for rays of understanding (and patience, above and beyond). My examiners Julian Bradfield and Ulrich Berger made for an unexpectedly enjoyable viva, and gave many valuable suggestions.

Thanks too to my friends, for distracting me when I needed to be distracted, and telling me to get on with it when I needed that too. In particular, Basheera Khan, Jo Gooch[1], Alfie Abdul-Rahman, Lidia Oshlyansky, Andy Gimblett, David Chisnall and Will Thimbleby have contributed much to my ongoing sanity over the years; and to the *labminton* players of the Fourth Floor Faraday – Teme Kahsai, Ben Spencer, Dave Arter – much appreciation: we've created something wonderful!

Finally, I wish to thank my parents, without whom nothing.

[1]Who knows good coffee when she drinks it.

# Contents

**List of Figures** **5**

**1 Introduction** **7**

1.1 Infinite state and infinitely-branching systems . . . 8
1.2 Labelled Transition Systems . . . 9
1.3 Equivalence relations on processes . . . 9
1.3.1 Silent actions . . . 11
1.4 Grammars as processes . . . 11
1.4.1 Mayr's Process Rewrite System hierarchy . . . 12
1.4.2 Basic Parallel Processes . . . 15
1.5 Approximant collapse . . . 16
1.6 Distance to disablings . . . 17
1.7 Organisation . . . 17

**2 Preliminaries** **19**

2.1 Ordinal Numbers . . . 19
2.1.1 Ordinal arithmetic . . . 20
2.1.2 Well-founded trees . . . 21
2.2 Cardinal numbers . . . 22
2.2.1 Regular cardinals . . . 23
2.3 Monoids . . . 23
2.3.1 Equivalence relations and congruences . . . 24
2.4 Semilinear sets . . . 25
2.4.1 Presburger arithmetic . . . 26
2.4.2 Finite automata . . . 27
2.5 Two machines . . . 29
2.5.1 Pushdown Automata . . . 29
2.5.2 Petri Nets . . . 30
2.5.3 Communication-free Petri Nets . . . 32

**3 Constructive Dickson's Lemma** **34**

3.1 Domination . . . 34

3.1.1 Domination-free trees . . . 35

3.2 Lexicographically ordered sequences . . . 36

3.3 A reification . . . 37

3.4 Further and related work . . . 39

**4 General Processes** **40**

4.1 Labelled Transition Systems . . . 40

4.1.1 Simple classes of general processes . . . 41

4.2 Equivalence relations on processes . . . 42

4.2.1 Trace and failures equivalence . . . 43

4.3 Bisimulation equivalence . . . 44

4.3.1 Bisimilarity on finite processes . . . 46

4.3.2 Bisimulation games . . . 46

4.3.3 Bisimulation approximants . . . 47

4.3.4 Semideciding $\not\sim$ on finitely-branching processes . . . 48

4.3.5 Finite approximants with finite alphabets . . . 49

4.4 Approximant hierarchy . . . 49

4.4.1 Optimal move trees . . . 50

4.4.2 Approximant hierarchy collapse . . . 51

4.5 Silent actions . . . 51

4.6 Other notions of weak bisimilarity . . . 54

**5 Processes from context-free grammars** **58**

5.1 GNF grammars as processes . . . 59

5.1.1 Basic Process Algebra . . . 59

5.1.2 Basic Parallel Processes . . . 60

5.1.3 Bisimilarity between BPA and BPP processes . . . 62

5.1.4 Unary languages . . . 63

5.1.5 PDA control-state bisimilarity hierarchy . . . 63

5.2 Extensions of context-free processes . . . 64

5.3 Deciding bisimilarity on context-free processes . . . 65

5.3.1 Caucal bases . . . 65

5.3.2 Hirshfeld trees . . . 66

5.3.3 Tableau proofs . . . 67

5.3.4 Jančar's distances-to-disablings . . . 68

5.4 Weak bisimilarity . . . 70

5.4.1 Totally normed BPP . . . 71

5.4.2 Purely-generated BPP . . . . . . . . . . . . . . . . . . . . . . 71
5.5 Weak bisimilarity is semidecidable on BPP . . . . . . . . . . . . . 71
5.6 The finite weak approximants . . . . . . . . . . . . . . . . . . . . . 73
5.7 Approximant collapse . . . . . . . . . . . . . . . . . . . . . . . . . 75
5.7.1 An $\omega^\omega$ bound for BPP . . . . . . . . . . . . . . . . . . . . . . 76

**6 Distances-to-disablings** **78**
6.1 Strong distances, and norm revisited . . . . . . . . . . . . . . . . . 78
6.2 $\mathrm{DD}^{\mathcal{C}}$ functions . . . . . . . . . . . . . . . . . . . . . . . . . . . 80
6.2.1 Approximant collapse . . . . . . . . . . . . . . . . . . . . . . . 83
6.2.2 The finite approximants . . . . . . . . . . . . . . . . . . . . . . 84
6.3 Weak distances . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 85
6.3.1 Refining weak distance . . . . . . . . . . . . . . . . . . . . . . 86
6.4 Extending $\mathrm{DD}_\tau$ . . . . . . . . . . . . . . . . . . . . . . . . . . . . 88
6.5 $\mathrm{DD}_\tau^{\mathcal{C},\mathcal{O}}$ . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 90

**7 $\mathrm{DD}_\tau$ on BPP** **92**
7.1 Difficulties . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 92
7.2 Computability of the $\mathrm{DD}_\tau$ functions . . . . . . . . . . . . . . . . . 93
7.2.1 Finite approximability . . . . . . . . . . . . . . . . . . . . . . . 95
7.2.2 Unary, non-norm-zero BPP . . . . . . . . . . . . . . . . . . . . 96
7.3 Insufficiency . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 96
7.4 $\mathrm{DD}'_\tau$ . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 98
7.5 $\mathrm{DD}_\tau^{\mathcal{O}}$, and its collapse . . . . . . . . . . . . . . . . . . . . . . . . 98

**8 Conclusions, and a research programme** **100**
8.1 A programme . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 101

**Bibliography** **103**

# List of Figures

1.1 Behaviourally distinct, trace-equivalent processes . . . . . . . . . . . 9
1.2 Van Glabbeek's linear/branching time hierarchy, [vG01] . . . . . . . 10
1.3 Two functionally equivalent, statically inequivalent processes . . . . 10
1.4 Chomsky hierarchy . . . . . . . . . . . . . . . . . . . . . . . . . . 12
1.5 A context-sensitive grammar, and its transition diagram . . . . . . 13
1.6 Chomsky Process hierarchy, [BCMS01] . . . . . . . . . . . . . . . . 14
1.7 Mayr's Process Rewrite System Hierarchy, . . . . . . . . . . . . . . 14
1.8 For every $n$, $M \overset{b}{\Rightarrow} C^n$, $M \overset{c}{\Rightarrow} MC^n$ and $M \overset{\epsilon}{\Rightarrow} MC^n$ . . . . . . . . . . . 16

2.1 A tree of height $\omega$ . . . . . . . . . . . . . . . . . . . . . . . . . . 21

4.1 A normed but non-well-founded LTS . . . . . . . . . . . . . . . . . . 41
4.2 $u \not\sim v$ . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 43
4.3 $v$ is capable of deadlocking, $u$ is not . . . . . . . . . . . . . . . . . 44
4.4 $s \equiv_L t$ . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 44
4.5 $u \equiv_f v$ . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 45
4.6 The Fan, $u \sim_\omega^! v$ . . . . . . . . . . . . . . . . . . . . . . . . . 47
4.7 The single-step weak approximants are not transitive . . . . . . . . . 53
4.8 $X \equiv_{\tau L} Y$ but $X \not\approx Y$ . . . . . . . . . . . . . . . . . . . . . 54
4.9 $u$ is finitely branching, $T(u)$ branches infinitely . . . . . . . . . . . 54
4.10 Two weakly bisimilar processes which are not branching bisimilar . . 55
4.11 Bisimilarity-like relations incorporating silent actions, [vGW96] . . . 56
4.12 Four approaches to silence (to be read left to right, top to bottom) . 57

5.1 A context-free grammar, and its transition system rooted at $X$ . . . 58
5.2 A context-free grammar in GNF . . . . . . . . . . . . . . . . . . . . 59
5.3 A $k$-control state PDA . . . . . . . . . . . . . . . . . . . . . . . . 64
5.4 Context-free processes $P, Q$ where $P \approx_\omega^! Q$ . . . . . . . . . . . . . 70

6.1 An illustration of $u \not\equiv_{\mathrm{DD}^{\aleph}} v \implies u \not\sim v$ . . . . . . . . . . . . . . . 81

6.2 $u \equiv_{\mathrm{DD}^c_{nw}} v$ . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 86
6.3 $u \approx^!_\omega v$, but $u \equiv_{\mathrm{DD}_\tau} v$ . . . . . . . . . . . . . . . . . . . . . . 88

7.1 $P \approx^!_\omega Q$ . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 92
7.2 $d(XX) = 1 > d(X) + d(X) = 0$ . . . . . . . . . . . . . . . . . . . 93
7.3 $X \equiv_{\mathrm{DD}_\tau} Y$ but $X \approx^!_\omega Y$ . . . . . . . . . . . . . . . . . . . . . . . 97

# Chapter 1

# Introduction

A system – be it a vending machine, a computer program or an aeroplane – can be described or specified at many levels of abstraction, just as a house can be presented as "tree bedrooms, detached", or a set of blueprints, or a model. One begins with the greatest abstraction, the briefest expression of what one wants, what is required; this is expanded, refined, the precise details of its implementation added, until one has an object far more solid than the original expression, but perhaps greatly more opaque. The question is: does this implementation – this model – meet our intentions? Is the process of refinement faithful? Travelling in the other direction, if we wish to prove that a device operates in a certain way, obeys a certain law (that a vending machine will always give the correct change; that a credit card system is secure from man-in-the-middle attacks; that a floating-point unit is correct), we can derive from it an idealised expression, an abstraction amenable to mathematical reasoning.

The level of abstraction this thesis is concerned with is to view a system as a (potentially infinite) collection of *states*, each representing one of its possible arrangements, and incorporating a notion of how it can move from one state to another. For example, one might model a clock as a set of times – its states being each hour:minute:second – together with an action *tick* which links successive moments. So, graphically, the node named 19:31:01 will have a single *tick*-labelled arrow connecting it to the node 19:31:02.

But we are not concerned with creating models per se (none of the examples in the thesis are "real world"), but the question of what happens when one wants to compare models already created: when one can say that *this* expression is essentially the same as *this* one (and the various meanings one can attach to "essentially the same").

## 1.1 Infinite state and infinitely-branching systems

Recent decades have seen a flourishing of research into the specification and verification of infinite-state systems. Automata invented to generate or recognise languages have been turned to use as models of behaviour, and novel machines, such as Petri Nets, developed; conceptions of equivalence have been imported (traces) and subsequently modified (failures), and new notions discovered: bisimilarity and its family (whose antecedents lie in logic). Process calculi have enabled infinite state, reactive, parallel and potentially non-terminating systems to be built, reasoned about, and validated [Mil88, BPS01]. The theory of sequential computation has a canonical model, the $\lambda$-calculus; concurrency still waits for unification. The schools of CSP (Hoare, [Hoa78a]), CCS (Milner, [Mil80, Mil89]) and ACP (Bergstra and Klop, [BK85]), each with a different emphasis – denotational or operational semantics, or geared towards equational reasoning – have produced rich familes of formalisms in which to define processes, so that, for instance, while it is in principle impossible to say whether an arbitrary program will halt, when defined within an appropriate framework its behaviour becomes both discoverable, and mechanically so.

Latterly, interest has alighted upon systems which may not only evolve into potentially infinitely many states, but can do so in a single step: infinitely branching systems, against which the usual methods for testing inequivalence fail, and for which new branches of theory are in the process of being grown. Our subject formalism is the Basic Parallel Processes, defined by context-free grammars in Greibach Normal Form, and found (by restriction) in ACP, CCS and Petri Net theory, which together with *weak bisimilarity* as a notion of equivalence defines infinite state, infinitely branching processes. While the decidability of *strong* bisimilarity (under which BPP processes branch finitely) has been met successfully, in its course giving rise to a formidable set of techniques – tableaux [BS90], Caucal bases [Cau90], Hirshfeld trees [CHM93], Jančar's DD functions [Jan03] – weak bisimilarity remains an open problem, solved only for restricted subclasses (the totally normed [Hir96], and normed purely generated BPP [Sti01c]) using approaches of tenuous applicability to the general case[1]. The main work of this thesis is the development of a technique based upon Jančar's DD functions, and inspired by his paper, with the potential – and with partial results to support its case – to settle the full problem positively.

[1]As is true on BPA, the sequential cousin of BPA, where Stříbrná and Cerná have attempted unsuccessfully to apply Hirshfeld trees to weak bisimilarity, [SČ02].

## 1.2 Labelled Transition Systems

Expressing the semantics of a process *denotationally* entails producing a function which maps processes to meanings, an approach which began with the work of Scott and Strachy (originally in terms of functions mapping input to output). (Structural) Operational semantics is an intuitive alternative, originating with Plotkin in [Plo81] ([Plo04]), in which the meaning of a program is exactly the steps it is able to perform: a directed labelled graph, whose nodes are the possible states the system can be in, and whose edges are labelled by the actions required to move from one state to another [AFV01]. Each path through the *labelled transition system* is then a possible execution run. For example, Figure 1.1 represents two processes, $v$ and $u$, one of which can perform an $a$ action and become a process capable of performing either a $b$ or a $c$ to get back to the original process, and the other able to choose between an $a$ transition to a process whose only enabled action is a $b$ back to $u$, or one whose sole action is a $c$ back to $u$.

$$\begin{tikzcd}
u \arrow[r, bend left, "a"] \arrow[d, bend left, "a"] & u_1 \arrow[l, bend left, "b"] & v \arrow[r, "a"] & v_1 \arrow[l, bend right, "c"'] \arrow[l, bend left=70, "b"] \\
u_2 \arrow[u, bend left, "c"] & & &
\end{tikzcd}$$

Figure 1.1: Behaviourally distinct, trace-equivalent processes

This offers a unified way to talk about processes: we can write $u \xrightarrow{a} v$ to mean the process $u$ performs an $a$ to become the process $v$, whether $u$ is modelled by, say, a Petri Net or CCS expression, and define equivalence relations independently of particular process calculi or automata.

## 1.3 Equivalence relations on processes

Van Glabbeek's Linear/Branching time hierarchy, Figure 1.2, presents twelve definitions of equivalence, arranged according to their coarseness, and topped by bisimilarity. At the bottom lies *trace equivalence*, the classical notion of equivalence from automata theory, and the least discriminating relation on processes. As noted above, each path traced through an LTS represents a potential run of the system, its sequence of labels is called a *trace*; two processes are trace equivalent when they produce identical sets of traces. No account is taken of branching structure; returning to Figure 1.1, $u$ and $v$ generate the same traces, but the first move of $u$ decides whether its second move can be a $b$ or a $c$, while $v$ can perform either: their behaviour differs.

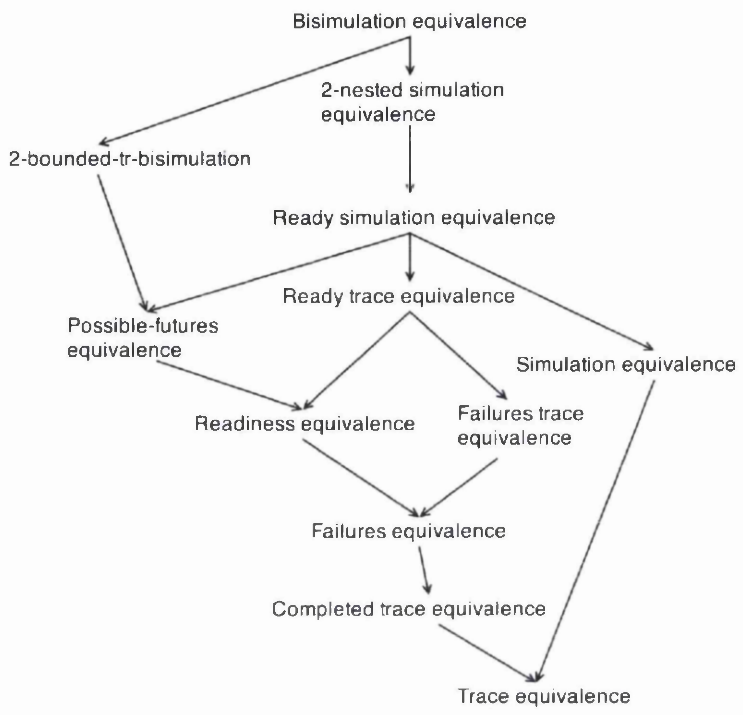

Bisimulation equivalence
2-nested simulation equivalence
2-bounded-tr-bisimulation
Ready simulation equivalence
Ready trace equivalence
Possible-futures equivalence
Simulation equivalence
Readiness equivalence
Failures trace equivalence
Failures equivalence
Completed trace equivalence
Trace equivalence

Figure 1.2: Van Glabbeek's linear/branching time hierarchy, [vG01]

Moving up the van Glabbeek hierarchy one finds an increasingly tight fit on what one would consider "behaviour" to be; *bisimilarity* ([Par81, Mil80]) is commonly referred to as the canonical notion of behavioural equivalence[2]. Processes are – coinductively – *bisimilar* when whatever one can do, the other can match, while maintaining bisimilarity.

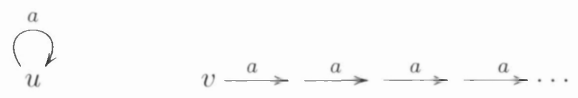


Figure 1.3: Two functionally equivalent, statically inequivalent processes

The processes of Figure 1.3 are bisimilar, as any $a$ from one can always be matched by an $a$ from the other; conversely, $u$ and $v$ of Figure 1.1 are not bisimilar, since a move of $u \xrightarrow{a} u_1$ can only be matched by $v \xrightarrow{a} v_1$, and $u_1$ is clearly not bisimilar to $v_1$: $v_1 \xrightarrow{c}$, while $u_1 \overset{c}{\not\rightarrow}$.

[2]For example, isomorphism (structural equivalence; equality up to the renaming of states) is stronger than bisimilarity, but distinguishes processes whose *behaviour* does not differ. In Figure 1.3 both $u$ and $v$ can do nothing more or less than an infinite sequence of $a$ actions; their transition systems differ structurally, but produce the same behaviour. (Though of course, when one comes to implement a system, whether it has one or infinitely many states is a difference one would not wish to overlook.)

Bisimilarity exhibits numerous pleasing properties (see e.g. [Sti98b]). It has a natural game theoretic formulation (§4.3.2); the property "this relation is a bisimulation" is expressible as a simple formula of first-order logic (Equation 5.27, page 72); if two processes are bisimilar, they will be considered equivalent under any interpretation (within van Glabbeek's hierarchy). Bisimilarity has been found to be tractable when all coarser notions are intractable, and often decidable where they are undecidable. On *finite state automata* trace equivalence is PSPACE-complete, while bisimilarity is decidable in $O(n \log n)$ time; it is a classical result that trace equivalence is undecidable on *pushdown automata* ([HU87]), yet in the past ten years bisimilarity has been proven decidable ([Sén98])[3].

### 1.3.1 Silent actions

To accommodate an idea of internal action, a silent or unobservable action name $\tau$ is introduced; a silent or *weak* transition involves a single observable action buffered before and after by any number of $\tau$-labelled transitions (§4.5),

$$u \overset{a}{\Rightarrow} v \quad =_{\text{def}} \quad u \xrightarrow{\tau^*} \xrightarrow{a} \xrightarrow{\tau^*} v \tag{1.1}$$

Every notion in the linear/branching time hierarchy of Figure 1.2 has its weak analogue, obtained by substituting $\rightarrow$ arrows by $\Rightarrow$. On finite transition systems this introduces no difficulties; on an infinite LTS, such as that generated by a pushdown automaton, CCS expression, BPP process, etc, it can be that one goes from having finitely many possibilities per transition to infinitely many.

## 1.4 Grammars as processes

Classically, grammars are used to generate languages – be they *natural languages* in the case of Chomsky [Cho56, Cho57][4] (and, long before him, Pāṇini [Ing67, Kak87]), the syntax of *programming languages* [Knu64]; or more abstract finite and infinite words, in the early-20th century work of Axel Thue [Thu14] and Emil Post [Pos43][5]. The motivating questions have been, respectively: understanding natural language (could English be described in terms of a context-free grammar, or something like

[3]The decidability of trace equivalence for *deterministic* pushdown automata was open for 30 years before Sénizergues' demanding paper [Sén97]; a proof a third the size was subsequently found by Stirling, [Sti01b], based on the observation that on deterministic processes, trace and bisimulation equivalence coincide (see §4.3).

[4]An example of a non-context free aspect of natural language is the crossing dependencies in subordinate clauses in Dutch.

[5]Published in 1943, but developed in the 20s.

it?); parsing computer programs (from *text* to *syntax*), [Joh75]; and the (Hilbert) programme of mechanising theorem proving (see e.g. [Boo87, MS05]).

| | Restriction | | Language type | Machine |
|---|---|---|---|---|
| Type 0 | $\alpha \to \beta$ | | Recursively enumerable | TM |
| Type 1 | $\alpha A\beta \to \alpha\gamma\beta$ | $\gamma \neq \epsilon$ | Context-sensitive | LB-TM |
| Type 2 | $A \to \gamma$ | $\gamma \neq \epsilon$ | Context-free | PDA |
| Type 3 | $A \to a\alpha$ | $\alpha \in V \cup \{\epsilon\}$ | Regular | FSA |

Figure 1.4: Chomsky hierarchy

The Chomsky hierarchy, Figure 1.4, divides grammars into four tiers, each with an elegant machine characterisation. We begin with a variable, and successively rewrite it according to the *transition rules*; our output is a string of *terminals*, a word (which, in the case of unrestricted grammars, can still be rewritten). A (standard) example, with variables $V = \{X, Y, C\}$ terminals $\Sigma = \{a, b, c\}$, and six transition rules is presented in Figure 1.5.

Their rôle as generators of *behaviour*, and specifically of labelled transition systems, is more recent, and begins with Caucal's *On the regular structure of prefix rewriting*, [Cau92] (see [Esp01] for an informal introduction). Terminals become *action names*; transition rules rewrite variables to variables,

$$\alpha \xrightarrow{a} \beta \qquad a \in \Sigma, \alpha, \beta \in V^* \tag{1.2}$$

As a further distinction, we can choose whether a state is a sequence or *multiset* of variables – whether it models a sequential or parallel system. In the *Chomsky process hierarchy*, Figure 1.6, unrestricted (Type 0) sequential grammars correspond to pushdown automata (Type 2 in the Chomsky hierarchy), while with parallel composition the machine equivalent is the Petri Nets. Type 2 processes are context-free grammars in Greibach Normal Form ([Gre65]); when interpreted sequentially we call them Basic Process Algebra (BPA) – PDA with a single control state and no $\epsilon$-transitions – while in parallel they are the Basic Parallel Processes. (For surveys on decidability questions on processes generating labelled transition systems, see [Mol96, BE97] and [Srb02b].)

### 1.4.1 Mayr's Process Rewrite System hierarchy

In his Process Rewrite System hierarchy, [May00b], Figure 1.7 (based on work by Moller in [Mol96]), Mayr generalises the Chomsky process hierarchy, allowing transitions to have forms that are sequential (S), parallel (P) and both (G), creating a

$$X \to abc, \quad X \to aYbc \qquad\qquad L(X) = \{a^n b^n c^n \mid n \geq 1\}$$
$$Y \to aYbC, \quad Y \to abC$$
$$Cb \to bC$$
$$Cc \to cc$$

$$\begin{tikzcd}
X \arrow[r] \arrow[d] & abc & & & \\
aYbc \arrow[r] \arrow[d] & aabCbc \arrow[r] & aabbCc \arrow[r] & aabbcc & \\
aaYbCbc \arrow[r] \arrow[d] \arrow[dddr] \arrow[dddddr] & aaabCbCbc \arrow[r] \arrow[ddr] & aaabCbbCc \arrow[r] \arrow[dr] & aaabCbbcc \arrow[r] & \cdots \\
\cdots & & & \cdots & \\
 & & aaabbCCbc \arrow[r] \arrow[dr] & \cdots & \\
 & aaaYbCbCbc \arrow[r] \arrow[d] \arrow[dr] \arrow[drr] & \cdots & \cdots & \\
 & \cdots & \cdots & \cdots & \\
 & \cdots & & &
\end{tikzcd}$$

Figure 1.5: A context-sensitive grammar, and its transition diagram

| | Restriction on $\alpha \xrightarrow{a} \beta$ | Restriction on $F$ | Parallel composition | Sequential composition |
|---|---|---|---|---|
| Type 0 | none | none | PDA | PN |
| Type $1\frac{1}{2}$ | $\alpha \in Q\Gamma$ | $F = Q$ | PDA | MSA |
| | $\beta \in Q\Gamma^*$ | | | |
| | $V = Q \uplus \Gamma$ | | | |
| Type 2 | $\alpha \in V$ | $F = \{\epsilon\}$ | BPA | BPP |
| Type 3 | $\alpha \in V$, $\beta \in V \cup \{\epsilon\}$ | $F = \{\epsilon\}$ | FSA | FSA |

Figure 1.6: Chomsky Process hierarchy, [BCMS01]

hierarchy that incorporates many of Chomsky's machine equivalents, and is strict with respect to bisimilarity[6].

$$1: \quad E ::= \epsilon \mid X \tag{1.3}$$

$$P: \quad E ::= \epsilon \mid X \mid E \| E \tag{1.4}$$

$$S: \quad E ::= \epsilon \mid X \mid E.E \tag{1.5}$$

$$G: \quad E ::= \epsilon \mid X \mid E \| E \mid E.E \tag{1.6}$$

$$\begin{tikzcd}
 & & \text{PRS } (G,G) \arrow[dl] \arrow[dr] & & \\
 & \text{PAD } (S,G) \arrow[dl] \arrow[dr] & & \text{PAN } (P,G) \arrow[dl] \arrow[dr] & \\
\text{PDA } (S,S) \arrow[dr] & & \text{PA } (1,G) \arrow[dl] \arrow[dr] & & \text{PN } (P,P) \arrow[dl] \\
 & \text{BPA } (1,S) \arrow[dr] & & \text{BPP } (1,P) \arrow[dl] & \\
 & & \text{FSA } (1,1) & &
\end{tikzcd}$$

Figure 1.7: Mayr's Process Rewrite System Hierarchy,

For the state of the art in decidability on process rewrite systems, Jiří Srba maintains the Roadmap of Infinite Results, [Srb02b][7]. The studied problems are strong and weak bisimilarity, strong and weak bisimilarity with finite state systems,

[6]The formalisms, as with pushdown automata and Petri Nets (§2.5.2)), fall short of full Turing power; reachability is decidable even for full PRS, [May00b]. For extensions with weak finite state units, see [KRS04]

[7]`http://www.brics.dk/~srba/roadmap/`

and strong and weak regularity (whether there exists a finite state process strongly or weakly bisimilar to a process in question), and results are given for each process rewrite system, and its *normed* subclass. A process is *normed* when no matter what sequence of transitions it performs, there is always a sequence which takes it to an "empty process" – if we (as in Equation 1.3) denote this process $\epsilon$, a process $\alpha$ is normed iff $\alpha \rightarrow^* \beta \implies \beta \rightarrow^* \epsilon$[8]. Many decidability results have often come first for a process class's normed subset, and afterward in full generality – bisimilarity on normed BPA was proved decidable in 1990, and for BPA in 1993. In 1999, bisimilarity was shown to be decidable on normed PA ([HJ99]); the general problem is still open. For both normed BPA and BPP there exist polynomial algorithms ([HJM96a] and [HJM96b, JK04] respectively). The only PRS for which bisimilarity is known to be decidable, but weak bisimilarity not, is PDA[9].

## 1.4.2 Basic Parallel Processes

A BPP is defined by a context-free grammar in Greibach Normal Form; its states are commutative sequences (or multisets) of variables; its transitions are given by the rule,

$$X \xrightarrow{a} \gamma \implies X\alpha \xrightarrow{a} \gamma\alpha \tag{1.7}$$

BPP represents a simple model of communication-free parallel computation, and was introduced by Christensen in his PhD thesis [Chr93]. (In the following example, $\epsilon$ denotes the empty process. The process $MC$ may either make an $a$-transition from its $M$ variable to $MCC$, a $b$-transition to $C$, or a $c$-transition from its $C$ variable to $M$.)

$$\begin{array}{lcl} M & \xrightarrow{a} & MC \\ M & \xrightarrow{b} & \epsilon \\ C & \xrightarrow{c} & \epsilon \end{array}$$

$$\begin{tikzcd}
M \arrow[r, bend left, "a"] \arrow[d, "b"] & MC \arrow[l, bend left, "c"] \arrow[r, bend left, "a"] \arrow[d, "b"] & MCC \arrow[l, bend left, "c"] \arrow[r, bend left, "a"] \arrow[d, "b"] & \cdots \arrow[l, bend left, "c"] \\
\epsilon & C \arrow[l, "c"'] & CC \arrow[l, "c"'] & \cdots \arrow[l, "c"']
\end{tikzcd}$$

Bisimilarity is PSPACE-complete on BPP [Jan03], while trace equivalence is undecidable [HJM96a]. Weak bisimilarity is believed to be decidable, but only partial results exist [Hir96, Stř99, Sti01c]. This aim of this thesis is to add a fourth reference to the preceding list.

[8]Note that this is not in general the same as saying that from every node in the LTS one can reach a node with no arrows leading from it. A Petri Net can have places marked but still be incapable of action. However, for BPA and BPP (communication-free Petri Nets), the notions do coincide.

[9]Note, on its parallel counterpart, PN, bisimilarity is already undecidable; bisimilarity might seem to favour sequential over parallel systems: on PAN and PRS it is undecidable, but for PAD the question is still open (this does not bode well, for the purposes of this thesis).

Non-bisimilarity is trivially semidecidable (each process has finitely many next states, so the state space can be explored until a mis-match is found); the research effort in proving it fully decidable centred on semideciding the positive case. Conversely, weak bisimilarity has been known to be semidecidable for a decade [Esp97], but no method for semideciding inequivalence has yet been found – the presence of silent actions allows a process to have infinitely many possible successor states (Figure 1.8); simple exploration is no longer possible.[10]

$$
\begin{aligned}
M &\xrightarrow{\tau} MC\\
M &\xrightarrow{\mathrm{b}} \epsilon\\
C &\xrightarrow{\mathrm{c}} \epsilon
\end{aligned}
$$

$$\begin{tikzcd}
M \arrow[r, bend left, "\tau"] \arrow[d, "\mathrm{b}"] & MC \arrow[l, bend left, "\mathrm{c}"] \arrow[r, bend left, "\tau"] \arrow[d, "\mathrm{b}"] & MCC \arrow[l, bend left, "\mathrm{c}"] \arrow[r, bend left, "\tau"] \arrow[d, "\mathrm{b}"] & \cdots \arrow[l, bend left, "\mathrm{c}"] \\
\epsilon & C \arrow[l, "\mathrm{c}"] & CC \arrow[l, "\mathrm{c}"] & \cdots \arrow[l, "\mathrm{c}"]
\end{tikzcd}$$

Figure 1.8: For every $n$, $M \overset{b}{\Rightarrow} C^n$, $M \overset{\mathrm{c}}{\Rightarrow} MC^n$ and $M \overset{\epsilon}{\Rightarrow} MC^n$

## 1.5 Approximant collapse

Weak bisimilarity is defined coinductively, but may be approached, and even reached, by its inductively defined *approximants* (§4.3.3). Game theoretically, these change the Defender's winning condition from survival for infinitely many turns to survival for $\kappa$ turns, for an ordinal $\kappa$, creating a hierarchy of relations successively closer to full weak bisimilarity. It can be seen that on any *set* of processes this approximant hierarchy collapses: there will always exist some $\kappa$ such that the $\kappa$th approximant coincides with weak bisimilarity (§4.4.2). One avenue towards the semidecidability of non-weak bisimilarity is the decidability of its approximants.

It is a long-standing conjecture that on BPP the weak approximant hierarchy collapses at $\omega \cdot 2$. If true, in order to semidecide inequivalence it would suffice to be able to decide the $\omega + n$ approximants. Again, there exist only limited results: the *finite* approximants are known to be decidable, but no progress has been made on the $\omega$th approximant, and thus far the best proven bound of collapse is $\omega_{1\mathrm{CK}}$ [Stř99] (the least non-recursive ordinal number). We significantly improve this bound to $\omega^{k \cdot 2}$ (§5.7; for a $k$-variable BPP), a key part of the proof being a novel constructive version of Dickson's Lemma [Dic13] (presented at CSL 2006, [HMS06]), covered in

[10]Dispiritingly, there is evidence that in sharp contrast to the situation between traces and strong bisimilarity, weak bisimilarity is *much* harder to decide than weak trace equivalence. For example, on processes with a finite state unit, weak (and strong) trace equivalence is $\Pi^0_1$-complete ([Jan95a]; the first level of the arithmetical hierarchy, [Rog67, Kec95]), yet weak bisimilarity is $\Sigma^1_1$-complete, [Srb03] (*highly undecidable*). Transplanted to BPP, this would at least suggest that weak bisimilarity is undecidable.

depth in Chapter 3.

## 1.6 Distance to disablings

The distances-to-disablings or DD functions were invented by Jančar to prove the PSPACE-completeness of bisimilarity on BPP [Jan03], and deployed subsequently to produce a $O(n^3)$ algorithm for its normed subset [JK04], and an algorithm for deciding bisimilarity between BPP and its sequential cousin, BPA [JKM03]. At the end of the paper he writes,

> ...the author conjectures that the method introduced here for strong bisimilarity will also turn out useful for showing decidability of weak bisimilarity for BPP

– a suggestion that has inspired the bulk of the work in this thesis. While the naïve weak analogue of the distances-to-disablings is easily seen to fail to capture weak bisimilarity (even on finite processes), a small modification suffices to express weak bisimilarity across a wide class of processes. In order to keep the functions finite we introduce further extensions, which are then applied to BPP.

## 1.7 Organisation

The thesis is organised as follows:

- Chapter 2 contains standard definitions and lemmas, chiefly on ordinal numbers, monoids, semilinear sets and Presburger Arithmetic.
- Chapter 3 concerns a constructive version of Dickson's Lemma (and is largely self-contained).
- Chapter 4 begins by defining a *general process* as a rooted, directed labelled graph, and defines and describes equivalence relations and approximant collapse with relation to them.
- Chapter 5 is on processes derived from context-free grammars. It reviews the current techniques for deciding bisimilarity and weak bisimilarity on BPP; the original work is a significant improvement on the bound of weak approximant collapse (making use of the work found in Chapter 3).
- Chapter 6 gives a generalisation of Jančar's DD functions, and further extend them to accommodate silent moves – the $\mathrm{DD}_\tau$ functions, and beyond, to the $\mathrm{DD}_\tau^{\mathcal{O}}$ functions.

- In Chapter 7 we apply the theory developed in Chapter 6 to the problem of weak bisimilarity on BPP. Although the greater problem remains unsolved, a number of partial results are found, and the thesis ends with:
- Chapter 8, conclusions, and a research programme.

# Chapter 2

# Preliminaries

Ordinal numbers, well-founded trees; monoids, semilinear sets and Presburger Arithmetic; finite and pushdown automata, and Petri Nets.

## 2.1 Ordinal Numbers

The counting numbers form an infinite sequence $0, 1, 2, \ldots$ which may nevertheless be extended: define $\omega$ to be the smallest number greater than every member of $\mathbb{N}$, and we can write,

$$0, 1, 2, \ldots, \omega, \omega + 1, \omega + 2, \ldots \tag{2.1}$$

and continue to $\omega \times 2$, $\omega \times 3$, and on $\omega^2$, $\omega^3$, and on,

$$\omega^\omega, \omega^{\omega^\omega}, \omega^{\omega^{\omega^\omega}}, \ldots, \epsilon_0, \epsilon_0 + 1, \ldots \tag{2.2}$$

(where $\epsilon_0$ is the smallest number $\kappa$ such that $\kappa = \kappa^\omega$). These are the *(transfinite) ordinal numbers*, denoted $\mathcal{O}$; and each is either a *successor ordinal* ($\kappa + 1$ for some ordinal $\kappa$), or a *limit ordinal.* All finite ordinals but 0 are successor ordinals, but $\omega$ is a limit ordinal as there is no $n$ s.t. $n + 1 = \omega$.

An intuition to the meaning of "a sequence of length $\kappa$", which will be useful when we come to consider *games* of an ordinal length §4.3.3, is to read $\omega$ as meaning "go arbitrarily far". $\omega \cdot 2$ is, then, an arbitrary distance, followed by an arbitrary distance; $\omega^2$ is an arbitrary number of arbitrary distances.

**Example 2.1 (arbitrarily far)** *Consider an aeroplane which can, at the start of its journey, take on board any amount of fuel. It is capable of flying any (finite) distance, so we say its potential range is $\omega$ kilometres. If allowed a single mid-air refuelling session, its potential range would be $\omega \cdot 2$; if permitted to decide, before take-off, on how many (finite) refuelling sessions it will have, the potential range* *becomes $\omega^2$.*

While each successor ordinal has a unique immediate predecessor, if $\lambda$ is a limit ordinal there is no greatest element smaller than it. Any strictly decreasing sequence of ordinals will be finite, allowing a transfinite form of induction. If, whenever a property is true of every ordinal less than $\kappa$ it must also be true of $\kappa$, we can conclude that the property holds of all ordinals.

We can make sense of this idea in terms of set theory using a realisation that owes to John von Neumann,

**Definition 2.1.1 (von Neumann's ordinals)** *A set $S$ is an* ordinal *iff*

1. *$R \subseteq S \implies R \in S$; and*
2. *$\subseteq$ is a total order on $S$, i.e. for $R, R' \subseteq S$, $R \subseteq R'$ or $R' \subseteq R$*

One can then represent the natural numbers as,

$$0 = \emptyset \tag{2.3}$$

$$n+1 = n \cup \{n\} \tag{2.4}$$

That is, $1 = \{\emptyset\} = \{0\}$, $2 = \{0,1\}$, $n = \{m \,|\, m < n\}$. These are ordinals, in the sense above, and extend naturally into the transfinite: $\omega =_{\text{def}} \{0, 1, \ldots\}$.

**Lemma 2.1.1** ($S \subset \mathcal{O} \implies \sup S \in \mathcal{O}$) *The supremum* $\sup S$ *of a set of ordinals $S$ is an ordinal.*

Ordinals can be uniquely expressed in base $\omega$, Cantor Normal Form:

**Theorem 2.1.1 (Cantor)** *For every ordinal $\kappa$ there exist unique sequences of natural numbers $c_1, \ldots, c_m$ and ordinals $\mu_1 > \ldots > \mu_m$ such that,*

$$\kappa = \omega^{\mu_1} c_1 + \omega^{\mu_2} c_2 + \ldots + \omega^{\mu_m} c_m$$

In particular, when each $\mu_i$ is a natural number we are able to express every ordinal less than $\omega^\omega$. This form is employed in support of Conjecture 5.7.1 (page 76).

### 2.1.1 Ordinal arithmetic

The usual recursive definitions of addition and multiplication on natural numbers extend to the ordinals:

$$\kappa + 0 =_{\text{def}} \kappa \tag{2.5}$$

$$\kappa + (\mu + 1) =_{\text{def}} (\kappa + 1) + \mu \tag{2.6}$$

$$\kappa + \lambda =_{\text{def}} \sup\{\kappa + \mu \,|\, \mu < \lambda\} \text{ where } \lambda \text{ is a limit ordinal} \tag{2.7}$$

and,

$$\kappa.0 =_{\text{def}} 0 \tag{2.8}$$

$$\kappa \cdot (\mu + 1) =_{\text{def}} (\kappa \cdot \mu) + \mu \tag{2.9}$$

$$\kappa \cdot \lambda =_{\text{def}} \sup\{\kappa \cdot \mu \mid \mu < \lambda\} \tag{2.10}$$

While addition and multiplication are associative, they need not commute: $1 + \omega = \sup\{1 + n \mid n < \omega\} = \omega < \omega + 1$; and $2 \cdot \omega = \sup\{2 \cdot n \mid n < \omega\} = \omega$.

### 2.1.2 Well-founded trees

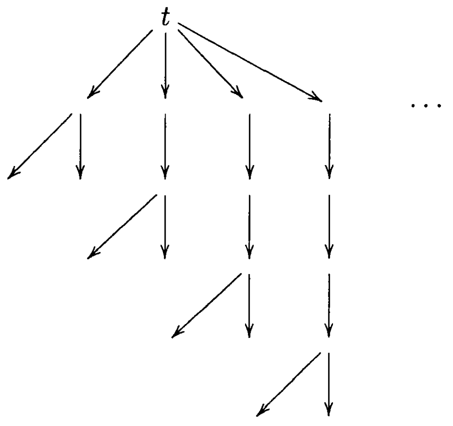


Figure 2.1: A tree of height $\omega$

A *tree* is a directed rooted graph in which there exists exactly one path (unique sequence of edges) from the root to each of its nodes[1]. A tree is is *well-founded* when no infinite paths exist within it: every sequence of steps that starts with the root and takes at each time a child to proceed with is finite. The *height* of a tree is defined,

**Definition 2.1.2 (h)** *If $t$ is the root of a tree, $h(t) =_{def} \sup\{h(s) + 1 \mid t \to s\}$. If $t$ is not well-founded, $h(t) =_{def} \infty$.* ■

Lemma 2.1.1 and the principle of well-founded recursion imply that if $t$ is well-founded, $h(t) \in \mathcal{O}$. Conversely, any ordinal $\kappa \in \mathcal{O}$ can (with transfinite induction) be recast as a well-founded tree of height $\kappa$:

1. The root $t$ is labelled $\kappa$;

[1]For example, Figure 2.1. Note that neither Figure 1.2 (page 10) nor Figure 4.10 (page 55) are trees, as their paths are not unique.

2. For each $\mu < \kappa$, add the well-founded tree which corresponds to $\mu$ as a child,

hence by hypothesis, $h(t) = \sup\{\mu \mid \mu < \kappa\} = \kappa$, and we find:

**Lemma 2.1.2 (ordinals and well-founded trees)** *$\kappa \in \mathcal{O}$ iff $h(t) = \kappa$ for some well-founded tree $t$.*

**Example 2.2 (addition of well-founded trees)** *Given two well-founded trees $s, t$, define $s + t$ to be the result of substituting every childless node of $t$ with $s$. The resultant operation is associative $(t + (s + u) = (t + s) + u)$, but not commutative. If $t_n$ is a tree of height $n$, and $t_\omega \rightarrow t_n$ for all $n$ (i.e., $h(t_\omega) = \omega$), then $h(t_\omega + t_1) = \omega + 1$, while $h(t_1 + t_\omega) = \omega$ (see §2.1.1).*

**Example 2.3 ($\mathcal{O}$ is a proper class)** *For each $\kappa \in \mathcal{O}$, Lemma 2.1.2 implies the existence of a tree $t_\kappa$ with $h(t_\kappa) = \kappa$; if $\mathcal{O}$ were a set, we could construct a tree $t$ with children $t \rightarrow s_\kappa$ for all $\kappa \in \mathcal{O}$. The same Lemma implies $\exists \mu \in \mathcal{O}.h(t) = \mu$, but by definition $h(t) \geq \mu + 1$.*

## 2.2 Cardinal numbers

Two sets have the same size or *cardinality* when their elements can be paired, one-to-one. $\omega + 1$ is a strictly larger ordinal than $\omega$, but when viewed as sets – as $\{0, 1, \ldots, \omega\}$ and $\{0, 1, \ldots\}$ respectively – they are of equal cardinality: $\omega$ maps to 0, 1 to 2, 2 to 3, etc. A *cardinal number* is an ordinal which can be used to describe the size of a set (i.e. any smaller ordinal must have a smaller cardinality). The first transfinite ordinal is $\omega$; when interpreted as a cardinal it is written $\aleph_0$. As $\mu$ and $\kappa$ range over the ordinals, $\aleph$ ranges over the cardinal numbers, and we will denote its class by $\mathcal{C}$.

On the finite numbers ordinality and cardinality coincide: $|n| = |\{0, 1, 2, \ldots, n - 1\}| = n$; into the transfinite, $|\omega| = |\mathbb{N}| = \aleph_0$, but

$$|\omega \cdot 2| = |\{0, 1, \ldots, \omega + 1, \omega + 2, \ldots\}| = |\mathbb{N} \cdot 2| = \aleph_0 \tag{2.11}$$

and even $|\epsilon_0| = \aleph_0$. The first uncountable ordinal is denoted $\omega_1$, $\omega_1 = \{\kappa \mid |\kappa| \leq \aleph_0\}$ – though we will touch here only briefly on numbers this large (§4.4.2, §5.7). The next largest cardinal from $\aleph_0$ is denoted $\aleph_1$; $\aleph_\omega = \sup\{\aleph_i \mid i < \omega\}$. The *successor* of a cardinal $\aleph$ is defined to be,

$$\text{succ}(\aleph) =_{\text{def}} |\inf\{\lambda \in \mathcal{O} \mid \aleph < |\lambda|\}| \tag{2.12}$$

(So, every successor cardinal is a limit ordinal.)

We use ordinals to measure the heights of trees, and cardinals to measure their widths. Graphs and trees may be classified according to their *branching-degree*: to what strict upperbound can be put on the number of children any node or vertex exhibits.

**Definition 2.2.1 (sub-$\aleph$-branching)** *A graph or tree is* sub-$\aleph$-branching *when the vertex degree of each of its nodes is bounded strictly above by* $\aleph$. ■

(See Definition 4.1.5 for the sub-$\aleph$-branching processes.)

**Example 2.4 (König's Lemma)**

*A well-founded tree $t$ is sub-$\aleph_0$-branching iff it is finite ([Kön36]).*

**Example 2.5**

*For every graph $G$ there exists an $\aleph \in C$ s.t. $G$ is sub-$\aleph$-branching. Specifically, denoting the vertex set of $G$ by $V(G)$, $G$ is a sub-succ$(|V(G)|)$-branching graph.*

### 2.2.1 Regular cardinals

$\aleph$ is a *regular cardinal* when it is greater than the supremum of any set of lesser cardinals fewer than $\aleph$. That is, any sequence of ordinals (each strictly less than $\aleph$) which converges on $\aleph$ must itself have at least $\aleph$ elements[2]. For example, $\aleph_0$ is regular, since it is greater than the supremum of any subset of $\aleph_0$, while $\aleph_\omega$ is by its definition not regular.

**Theorem 2.2.1 (sub-regular cardinal-branching trees)**

*If $t$ is a sub-$\aleph$-branching tree, for a regular $\aleph$, then $h(t) < \aleph$.*

**Proof:** An induction on $h(t)$. Imagine that $t \rightarrow s \implies h(s) < \aleph$. Let $A = \{h(s)+1 \mid t \rightarrow s\}$, then $|A| < \aleph$, and $h(t) = \sup A < \aleph$. □

We use regular cardinals to simplify several proofs; it is useful to observe,

**Lemma 2.2.1 (regular cardinals)** *For every cardinal there exists a regular cardinal greater than it. In particular (assuming the Axiom of Choice), the* successor *of any infinite cardinal is regular, [End77].*

## 2.3 Monoids

**Definition 2.3.1 (monoid)** *A* monoid $(S,+)$ *is a set $S$ with an associative binary operation $+ : S \times S \rightarrow S$ and identity element $\mathit{0} \in S$ ($\forall s \in S. s + \mathit{0} = \mathit{0} + s = s$). It is* commutative *if $\forall s,t \in S. s+t = t+s$.* ■

[2] A cardinal is either finite, regular or *singular*.

**Example 2.6 (words over $\Sigma$)** *Let $\Sigma^*$ denote the set of sequences over $\Sigma$, with $\epsilon$ as the empty sequence, and $\cdot$ be a concatenation function (so $\Sigma^*$ is better expressed as the closure of $\Sigma$ with $\cdot$), then $(\Sigma^*, \cdot)$ is a monoid with identity $\epsilon$.*

A *vector* or $k$-tuple is sequence of natural numbers of fixed length,

$$\vec{x} = (x_1, x_2, \ldots, x_k) \in \mathbb{N}^k \tag{2.13}$$

We refer to $x_1, \ldots, x_k$ as its *components*, and write,

$$\vec{x}(i) =_{\text{def}} x_i \tag{2.14}$$

The *free commutative monoid* over a set $V = \{X_1, \ldots, X_k\}$ is $(V^{\otimes}, \cup)$, where $V^{\otimes}$ represents the set of multisets over $V$, and $\cup$ is multiset union. This is essentially the same as a recurring monoid, the $k$-dimensional vector space $(\mathbb{N}^k, +)$ (where $+$ is pointwise addition), in that they are related by the *monoid isomorphism*,

$$f(x_1, x_2, \ldots, x_k) =_{\text{def}} X_1^{x_1} \ldots X_k^{x_k} \tag{2.15}$$

(using monomial notation to express multisets). A useful monoid *homomorphism* is the *Parikh mapping* from sequences (over $\Sigma$) to vectors (of size $|\Sigma|$):

**Definition 2.3.2 (parikh mapping)** *If $w \in \{a_1, \ldots, a_m\}^*$*

$$\vec{w} =_{def} (w(a_1), \ldots, w(a_m)) \tag{2.16}$$

*where, $w(a_i) =_{def}$ the number of $a_i$s in $w$.* ∎

**Example 2.7 (language)** *A* language *is a subset of words over an alphabet $\Sigma$, $L \subseteq \Sigma^*$; a* commutative language *is the parikh mapping of a language $L$, $\{\vec{w} \mid w \in L\}$, i.e. a subset of $\mathbb{N}^{|\Sigma|}$.*

### 2.3.1 Equivalence relations and congruences

**Definition 2.3.3 (equivalence relations)** *A binary relation $R$ on the monoid $(S, +)$ is an* equivalence relation *when it satisfies, for all $u, v, s \in S$,*

1. *$uRu$ (identity);*
2. *if $uRv$ then $vRu$ (symmetry); and*
3. *if $uRv$ and $vRs$ then $uRs$ (transitivity).*

*if, moreover, it satisfies $\forall u, v, s. uRv \implies u + sRv + s$ we call $R$ a* congruence. ∎

**Example 2.8 (sum equivalence)** *On the monoid* $(\mathbb{N}^k, +)$, *define two vectors* $\vec{x}, \vec{y}$ *to be* sum-equivalent, $\vec{x} \equiv_S \vec{y}$, *when* $\sum_{i=1}^{k} \vec{x}(i) = \sum_{i=1}^{k} \vec{y}(i)$. $\equiv_S$ *is an equivalence relation, and in fact is a congruence, since* $\vec{x} \equiv_S \vec{y}$ *implies, for any* $\vec{z}$, *that* $\vec{x} + \vec{z} \equiv_S \vec{y} + \vec{z}$.

Extending Example 2.6, a monoid $(S, +)$ is *generated* by $S' \subseteq S$ when $S$ equals the closure of $S'$ under $+$, $S'^* = S$. Every monoid is generated by itself; the *rank* of a monoid is the cardinality of its smallest set of generators.

**Example 2.9** $((\mathbb{N}^k, \times)$ **and** $(\mathbb{N}^k, +))$

1. $(\mathbb{N}^k, \times)$, *the k-tuples with pointwise multiplication, is not finitely generated. Consider any finite set of vectors* $M \subseteq \mathbb{N}^k$, *and let p be a prime number larger than any component of any element of* $M$, *then* $\mathbb{N}^k \ni (p, 0, \ldots, 0) \notin M^*$.

2. $(\mathbb{N}^k, +)$ *is finitely generated, and of rank k. Let,*

$$\vec{0}^i(j) =_{def} \begin{cases} 1 & \leftarrow i = j \\ 0 & \leftarrow i \neq j \end{cases} \tag{2.17}$$

*i.e.* $\vec{0}^1 = (1, 0, \ldots, 0)$, $\vec{0}^k = (0, \ldots, 0, 1)$, *then* $\mathbb{N}^k$ *equals the closure of* $\{\vec{0}^1, \ldots, \vec{0}^k\}$ *under pointwise addition.*

**Theorem 2.3.1 (Redei, [Red65, Fre68])** *If* $R \subseteq S \times S$ *is a congruence on a finitely generated commutative monoid* $(S, +)$ *then* $R$ *is finitely generated*[3].

**Example 2.10 (finitely generated sum congruence)** *By Theorem 2.3.1 the congruence* $\equiv_S$ *of Example 2.8 is finitely generated. Indeed,*

$$\equiv_S = \left\{ (\vec{0}^i, \vec{0}^j)_{1 \leq i,j \leq k} \right\}^* \tag{2.18}$$

A concept we will return to with Caucal Bases in §5.3.1 is that of a *congruence base* (or *Thue congruence*),

**Definition 2.3.4** ($\overset{R}{\equiv}$) *If* $R$ *is a binary relation on* $(S, +)$, $\overset{R}{\equiv}$ *is the least congruence containing* $R$. *That is,* $u \overset{R}{\equiv} v \wedge u' \overset{R}{\equiv} v' \implies u + u' \overset{R}{\equiv} v + v'$. ■

## 2.4 Semilinear sets

A set $A \subseteq \mathbb{N}^k$ is *linear* iff there is a finite sequence of vectors (its base) $\vec{x}, \vec{x}_1, \ldots, \vec{x}_m \in \mathbb{N}^k$ with $A = \{\vec{x} + \vec{x}_1 n_1 + \ldots + \vec{x}_m n_m \mid n_1, \ldots, n_m \in \mathbb{N}\}$. For example, that $(\mathbb{N}^k, +)$

[3]This holds even if $(S, +)$ does not have an identity element.

is finitely generated is enough to show that $\mathbb{N}^k$ is linear. The *semilinear sets* are the finite unions of linear sets; the *base* of a semilinear set is the collection of bases of its linear constituents.

**Theorem 2.4.1 (Eilenberg and Schützenberger, [ES69])** *If $R \subseteq \mathbb{N}^k \times \mathbb{N}^k$ is a congruence on a finitely generated commutative monoid $(\mathbb{N}^k, +)$ then $R$ is semilinear.*

**Example 2.11 (semilinearity of $\equiv_S$)** *$\equiv_S$ (Example 2.10) is in fact* linear,

$$\equiv_S = \left\{ (\vec{0}, \vec{0}) + \sum_{1 \leq i,j \leq k} (\vec{0^i}, \vec{0^j}).n_{i,j} \mid n_{i,j} \in \mathbb{N} \right\} \tag{2.19}$$

The importance of this result rests on the intimate connection between the semilinear sets and *Presburger arithmetic*, the first-order theory of addition.

### 2.4.1 Presburger arithmetic

*Presburger Arithmetic* is the first-order theory of addition: first order logic over $\mathbb{N} = (\mathbb{N}, + <)$. By $\phi(x_1, \ldots, x_k)$ we mean a formula with free variables $x_1, \ldots, x_k$. If there exist instantiations $a_1, \ldots, a_k \in \mathbb{N}^k$ for the variables which make the formula true, we write, $\phi[a_1, \ldots, a_k]$. The set of satisfying tuples is denoted,

$$[[\phi]] =_{\text{def}} \{(a_1, \ldots, a_k) \in \mathbb{N}^k \mid \phi[a_1, \ldots, a_k]\} \tag{2.20}$$

Unlike Peano Arithmetic (the first-order theory of addition and multiplication),

**Theorem 2.4.2 (Presburger, [Pre29, Tar51, FR74])** *Presburger Arithmetic is decidable in 2-EXP time (and any decision procedure must have at least a double-exponential worst-case run time).*

A set $A \subseteq \mathbb{N}^k$ is *Presburger* when there exists a formula of $k$ free variables $\phi(x_1, \ldots, x_k)$ with $A = [[\phi]]$. A relation $R$ is *Presburger computable* if and only if it a Presburger set.

**Theorem 2.4.3 (Ginsberg and Spanier, [GS66])** *A set $A \in \mathbb{N}^k$ is semilinear if and only if it is Presburger; and, each description is effectively calculable from the other.*

An easy consequence of this is,

**Lemma 2.4.1 (semilinear set membership)** *Semilinear-set membership is decidable.*

**Example 2.12 (sum equivalence)** *To continue the thread of Example 2.11, Theorem 2.4.3 implies that $\equiv_S$ is Presburger. Easily, let $\phi(x_1, \ldots, x_k, y_1, \ldots, y_k) \equiv x_1 + \ldots + x_k = y_1 + \ldots + y_k$, then*

$$\equiv_S = [[\phi]] \tag{2.21}$$

**Corollary 2.4.1 (congruences on $(\mathbb{N}^k, +)$)** *If $R$ is a congruence on $(\mathbb{N}^k, +)$,*

1. *It has a finite representation as the base of a semilinear set;*
2. *If this can be found, it is possible to produce a statement of Presburger Arithmetic $\phi_R$ which expresses $R$, and thereby $R$ is decidable.*

### 2.4.2 Finite automata

A *finite automaton* is a finite directed graph $\mathcal{A} = (V, \Sigma, \rightarrow, u, F)$ with vertices $V$, start-state $u \in V$, and *final states* $F \subseteq V$, whose edges are labelled from $\Sigma$ [JJ79]. It recognises a word $w \in \Sigma^*$ exactly when there exists a sequence of edges from $u$ to a state in $F$, whose labels equal $w$. The *language* of a finite automaton is, then,

$$L(\mathcal{A}) =_{\text{def}} \{a_1 \ldots a_n \mid u \xrightarrow{a_1} \ldots \xrightarrow{a_n} u' \in F\} \tag{2.22}$$

while its *commutative language* (Example 2.7) is $LC(\mathcal{A}) =_{\text{def}} \{\vec{w} \mid w \in L(\mathcal{A})\}$. Here we find a further connection to the semilinear sets. First, define an inverse Parikh mapping (where $\Sigma = \{a_1, \ldots, a_m\}$),

$$\text{word}(x_1, \ldots, x_m) =_{\text{def}} a_1^{x_1} \ldots a_m^{x_m} \tag{2.23}$$

then given a semilinear set,

$$\begin{aligned} S &= \{\vec{x}_{1,0} + \vec{x}_{1,1}n_1 + \ldots + \vec{x}_{1,a_1}n_{a_1} \mid n_1, \ldots, n_{a_1} \in \mathbb{N}\} \cup && (2.24) \\ & \quad \ldots && (2.25) \\ & \quad \{\vec{x}_{l,0} + \vec{x}_{l,1}n_1 + \ldots + \vec{x}_{l,a_1}n_{a_l} \mid n_1, \ldots, n_{a_l} \in \mathbb{N}\} && (2.26) \end{aligned}$$

it is easy to construct an automaton $\mathcal{A}$ with $LC(\mathcal{A}) = S$. Namely, for each linear set $\{\vec{x}_{i,0} + \vec{x}_{i,1}n_1 + \ldots + \vec{x}_{i,a_1}n_{a_i} \mid n_1, \ldots, n_{a_1} \in \mathbb{N}\}$ draw a word$(\vec{x}_{i,0})$-labelled sequence of arrows from $u$ to a final state $f_i \in F$, and add to $f_i$, for each $\vec{x}_{i,j}$, $j \geq 1$, word$(\vec{x}_{i,j})$-labelled self-loops. I.e. we generate $\vec{x}_{i,0}$, then allow any combination of $\vec{x}_{i,j}$ $(j > 0)$ to be added.

**Example 2.13 ($\equiv_S$ as a finite automaton)** *Since $\equiv_S$ is a set of $k \times 2$-tuples, we can represent it as the following $2k^2 + 1$-state, $4k^2$-edged finite automaton over the* *language* $\Sigma = \{a_1, \ldots, a_k, b_1, \ldots, b_k\}$,

$$\begin{tikzcd} u \arrow[loop left, "b_j", "a_i"'] \arrow[loop right, "b_j", "a_i"'] \end{tikzcd} \qquad \forall 1 \leq i, j \leq k$$

*with* $F = \{u\}$.

For the reverse direction, given an automaton $\mathcal{A}$ over $\Sigma = \{a_1, \ldots, a_m\}$ we wish to produce (the base of) a semilinear set $S \subseteq \mathbb{N}^m$ s.t. $LC(\mathcal{A}) = S$. To this end, we define the *pumping paths* and *pure pumping paths* of $\mathcal{A}$.

A *pumping path* from a node $v \in V$ is a sequence of edges $v \xrightarrow{a_1} \xrightarrow{a_2} \ldots \xrightarrow{a_l} v$; a pumping path is *pure* when it contains no other pumping paths. Clearly, for a $k$ state automaton $\mathcal{A}$, any sequence of $k$ or more edges must contain a pumping path, and the number of pure pumping paths is finite. We observe, for an automaton $\mathcal{A}$ with start-state $u$,

1. If $v$ has a pumping path $v \xrightarrow{b_1} \ldots \xrightarrow{b_m} v$, and there is a sequence of edges $u \xrightarrow{a_1} \ldots \xrightarrow{a_l} v$, the commutative language of $\mathcal{A}$ contains the linear set

$$\{a_1 \,\vec{\ldots}\, a_l + b_1 \,\vec{\ldots}\, b_m . n \mid n \in \mathbb{N}\} \tag{2.27}$$

2. If $v$'s pumping path is not pure, i.e.

$$\exists t. v \xrightarrow{b_1} \ldots \xrightarrow{b_n} t \xrightarrow{b_{n+1}} \ldots \xrightarrow{b_{n+m}} t \xrightarrow{b_{n+l+1}} \ldots \xrightarrow{b_m} v \tag{2.28}$$

then we can widen the above linear set to,

$$\begin{aligned} \{\vec{(}a_1 \ldots a_l) \quad &+ \quad b_1 \ldots b_n \vec{b}_{n+l+1} \ldots b_m . n_1 && (2.29)\\ &+ \quad b_{n+1} \,\vec{\ldots}\, b_{n+l} . n_2 && (2.30)\\ &\quad\ \mid n_1, n_2 \in \mathbb{N}\} \subseteq LC(\mathcal{A}) && (2.31) \end{aligned}$$

3. In such a way, every sequence rooted at $u$ can be decomposed into a path, followed by a series of pure pumping paths.

Since there are finitely many pure pumping paths, bounded in the size of the automaton, we are able to effectively compute the bases of a semilinear representation of its commutative language. That is to say:

**Theorem 2.4.4 (semilinear sets and finite automata)** *A set $S \in \mathbb{N}^k$ is semilinear if and only it is recognised as the commutative language of a finite automaton (and the translations between the two representations are effective).*

The *star height* of a language is a measure of how many nested loops are required to generate it (in fact, the number of nested Kleene-star operators when given as a regular expression). It is a classical result that there are languages of arbitrary star height [Coh70]. When looking at commutative languages, a corollary of the construction used in Theorem 2.4.4 gives us:

**Corollary 2.4.2 (commutative regular language star height)** *The star-height of a commutative finite automaton language is at most 1.*

## 2.5 Two machines

We have already seen one class of automata, the finite state machines, §2.4.2. Here we give two more, each with a connection to the Basic Parallel Processes §5.1.2.

### 2.5.1 Pushdown Automata

*Pushdown automata* add to the finite automata of §2.4.2 an unbounded stack. A pushdown automaton is a quintuple, $(P, \Gamma, \Sigma, \rightarrow, Z, p)$ of,

1. Control states $P$;
2. Stack symbols $\Gamma$;
3. Initial stack symbol $Z \in \Gamma$;
4. Initial state $p \in P$; and
5. Transition relation, $P \times \Gamma \times \Sigma \rightarrow P \times \Gamma^*$

A *position* is a pair of control state and stack (note that this would be our usual meaning of "state"), $(q, \alpha)$ $(\alpha \in \Gamma^*)$. If in position $(q, X\alpha)$, a rule $(q, X)\xrightarrow{a}(q', \beta)$ means the automaton can *read* an $a$ and move to position $(q', \beta\alpha)$; we say that $a$ has been *accepted.* This notion extends to sequences of letters; the *language* of a PDA is the set of words it accepts. These automata can evolve into potentially infinitely many positions; their languages – the *context-free languages* – are of a strictly greater complexity than those given by the finite automata [JJ79].

The automata's power as language generators does not increase with the number of control states; the single-state PDA already generate all of the context-free languages. As *process generators* this does not hold (see Lemma 5.1.2): increasing the number of control states widens the class of processes which may be described. Single-control-state PDAs, as used to define processes, are called Basic Process Algebra, and represents the sequential cousin of the Basic Parallel Processes.

### 2.5.2 Petri Nets

The first well-studied theory of concurrency is a class of automata called the Petri Nets, invented by Carl Adam Petri in his PhD thesis, [Pet62] (see e.g. [Rei85, EN94], of the extensive literature available[4]). They are able to model *true concurrency* – the actions of a pushdown automata can be totally ordered by time, but for a Petri Net, time can be made a partial order[5]. (However, it is only the *interleaving semantics*, not the *concurrency semantics* of Petri Nets which interests us here, as we wish to use them to generate Labelled Transition Systems, in which every action is totally ordered by time.)

A *net* $(P, Tr, \Sigma, l, W)$ is a finite collection of *places*, $P$, each holding some (finite, unbounded) number of tokens, and connected by *transitions*, $Tr$, labelled with action names, $l : Tr \to \Sigma$. The *weight* function $W$ connects places and transitions[6]:

$$W : (P \times Tr) \cup (Tr \times P) \to \mathbb{N} \tag{2.32}$$

A transition can *fire* when all of its input places contain at least one token; it then removes one token from each of its inputs, and adds tokens to each of its outputs. A *state* is a vector of natural numbers: the number of tokens in each place, and in this Petri Nets are an example of $k$-tuple processes (§4.1.1).

**Example 2.14 (a Petri Net)** *(This example comes from [BCMS01].)*

*Define a net $\mathcal{N} = (P, Tr, \Sigma, l, W)$, with places $P = \{p, q, r, s\}$, transitions $Tr = \{t_1, t_2, t_3, t_4, t_5, t_6\}$, actions $\Sigma = \{a, b, c, d\}$, labels $l(t_1) = l(t_6) = a$, $l(t_2) = l(t_3) = b$, $l(t_4) = c$, $l(t_5) = d$, and weight function,*

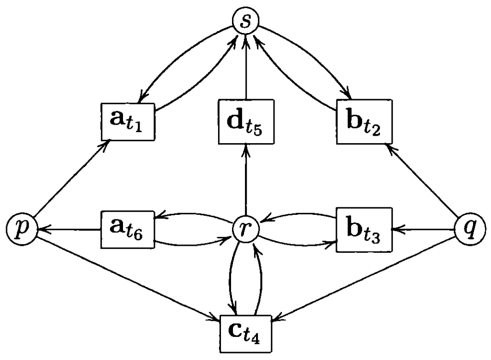
$$\begin{array}{lll}
W(p,t_1) = 1 & W(p,t_4) = 1 & \\
W(r,t_6) = 1 & W(r,t_3) = 1 & W(r,t_4) = 1 \\
W(q,t_2) = 1 & W(q,t_4) = 1 & \\
W(s,t_1) = 1 & W(s,t_2) = 1 & \\
W(t_1,s) = 1 & W(t_2,s) = 1 & \\
W(t_3,r) = 1 & W(t_3,q) = 1 & W(t_4,r) = 1 \\
W(t_5,s) = 1 & W(t_6,q) = 1 & W(t_6,r) = 1
\end{array}$$

[4]*Petri Nets World*, `http://www.informatik.uni-hamburg.de/TGI/PetriNets/`

[5]There is a disparity between the way a physicist sees a system (which might be as a set of interacting particles, with no notion of a global clock), and how it is seen by computer science (a next-state function, stepping through time). Petri Nets are an attempt to bridge this divide.

[6]Petri Nets can be viewed as a generalisation of the *Vector Addition Systems* of [KM69] – the reachability sets of an $n$-place feedback-free Petri Net (that is, whose transitions do not have self-loops) is isomorphic to an $n$-dimensional VAS. The ($n$-dimensional) *Vector Addition Systems with States*, [HP79], are essentially Petri Nets with no more than $n$ unbounded places.

*(with W being 0 on all other place/transition pairs). Though cumbersome to express in terms of functions and sets, the key to Petri Nets is their graphical representation:*



*(Places are circled, transitions boxed.) If r is given one token, the set of possible firing sequences – the language generated – happens not to be context-free.*

Petri Nets are not Turing complete – the *reachability problem* (whether there exists a sequence of transitions between two given states, analogous to the Halting problem) is decidable [EN94][7] – but become so if *inhibitor arcs* are included (transitions which fire when their input places are unmarked[8], i.e. tests for zero). Similarly, pushdown automata are not Turing complete – there, the addition of a second stack is enough to allow them to compute everything that is computable – but although *bisimilarity* (§4.3) is decidable for PDAs ([Sén98]), on Petri Nets it is already undecidable ([Jan95b]); *weak bisimilarity* (Definition 4.5.1) is undecidable for Petri Nets with even a single unbounded place ([May03a]; it is unknown whether bisimilarity is decidable on such nets), as is true of PDAs with a single stack symbol (in addition to a bottom-of-stack symbol which cannot be removed; *ibid*).

A *marking* $M$ is a tuple of natural numbers expressing the number of tokens on each place, $M \in \mathbb{N}^{|P|}$ (so, if $p \in P$, $M(p)$ equals the number of tokens on $p$); a Petri Net *process* is a pair of net and marking, $(\mathcal{N}, M)$. We write $M \xrightarrow{a} M'$ when a net marked from $M$ has an $a$-labelled transition enabled, which when fired produces a marking of $M'$. That is,

$$M \xrightarrow{a} M' \quad \text{iff} \quad \begin{array}{l} \exists t \in Tr.l(t) = a \wedge \forall p \in P.W(p,t) > 0 \implies M(p) > 0 \text{ and} \\ \forall p \in P.M(p) - W(p,t) + W(t,p) = M'(p) \end{array} \tag{2.33}$$

Analogously, we write $M \xrightarrow{t} M'$, for a $t \in Tr$, to mean $t$ is enabled in $M$, and produces from it $M'$ when fired. A standard result from Petri Net theory is,

[7]On PAN (Figure 1.7), the least generalisation of Petri Nets and PA, reachability remains decidable, [May97].

[8]It has recently been found that reachability is decidable for nets with a single inhibitor arc, [BLM89, Rei04].

**Lemma 2.5.1 (Petri Net transition sequences)** *For $\sigma \in Tr^*$,*

$$M \xrightarrow{\sigma} M' \quad \Longrightarrow \quad \forall p \in P. M(p) + \sum_{t \in Tr} (W(t,p) - W(p,t)) \cdot \vec{\sigma}(t) = M'(p)$$

### 2.5.3 Communication-free Petri Nets

A net is *communication free* when its transitions have exactly one input place a piece; we cannot synchronise places, cannot say "if this place *and* this place are enabled, fire" (analogous to the divide between context-sensitive and context-free grammars).

**Definition 2.5.1 (communication-free Petri Nets)**
*A Petri Net $\mathcal{N} = (P, Tr, \Sigma, l, W)$ is* communication free *when, for $p, p' \in P, t \in Tr$, $W(p,t) > 0 \wedge W(p',t) > 0 \implies p = p'$.* ■

Graphically, each box has exactly one arrow entering it.[9] A place $p$ is *markable* in $\mathcal{N}$ from $M$ if there exists a transition sequence the result of which puts a token in $p$. The *subnet* $\mathcal{N}_\sigma$ generated by a sequence of transitions $\sigma \in Tr^*$ has as transitions, $Tr_\sigma = \{\sigma_i\}$ and places,

$$P_\sigma =_{\text{def}} \{p \in P \mid \exists i. W(p, \sigma_i) > 0 \text{ or } W(\sigma_i, p) > 0\} \tag{2.34}$$

For the communication-free nets, Esparza has found a stronger version of Lemma 2.5.1 ([Esp97]):

**Lemma 2.5.2 (communication-free Petri Net transition sequences)**
*For $\sigma \in Tr^*$, $M \xrightarrow{\sigma}$ if and only if,*

1. *$\forall p \in P$, $M(p) + \sum_{t \in Tr}(W(t,p) - W(p,t)).\vec{\sigma}(t) \geq 0$; and*
2. *every $p \in P_\sigma$ is markable from $M$ in the subnet generated from $\sigma$.*

And goes on to prove that,

**Lemma 2.5.3 (Esparza's Reach)** *Reach $=_{def} \{(M, \vec{\sigma}, M') \mid M \xrightarrow{\sigma} M'\}$ is effectively semilinear.*

[9] A useful shorthand, employed in Chapter 5, in defining transitions is $X \xrightarrow{a} \alpha$, where $X$ is a place, $a$ the transition's action, and $\alpha$ a tuple of each place (possibly repeated) to which the transition outputs tokens. The net can be described as a commutative context-free grammar in Greibach normal form – it is, in fact, exactly a BPP, §5.1.2. (The connections between commutative grammars and Petri Nets have long been of interest to researchers – see for example [CRM74].)

**Proof:** For $P' \subseteq P$ and $T' \subseteq Tr$, define sets of markings and Parikh-mapped transition sequences restricted to $P'$ and $T'$ respectively,

$$\begin{aligned} \mathcal{M}(P') &=_{\text{def}} \{M \mid M(p) > 0 \text{ iff } p \in P'\} \\ \mathcal{T}(T') &=_{\text{def}} \{\vec{\sigma} \mid \vec{\sigma}(t) > 0 \text{ iff } t \in T'\} \end{aligned}$$

Define, $\text{Reach}(P', T') =_{\text{def}} \{(M, \vec{\sigma}, M') \mid M \in \mathcal{M}(P'), \vec{\sigma} \in \mathcal{T}(T') \text{ and } M \xrightarrow{\sigma'} M'\}$, and find that:

$$\text{Reach} = \bigcup_{P' \subseteq P, T' \subseteq Tr} \text{Reach}(P', T') \tag{2.35}$$

Since semilinearity is preserved by finite union, it suffices to prove that each set $\text{Reach}(P', T')$ is effectively semilinear. Note that if a place is markable from some marking in $\mathcal{M}(P')$, it must be markable from every marking in that set. Consider some $\vec{\sigma} \in T'$, and the subnet $\mathcal{N}_\sigma$ it generates. There are now two cases:

1. Every place of $\mathcal{N}_\sigma$ is markable from $\mathcal{M}(P')$. Applying Lemma 2.5.3, we find that $\text{Reach}(P', T')$ equals the solutions $(M, \vec{\sigma}, M')$, for all $p \in P$, to,

$$M(p) = M'(p) + \sum_{t \in Tr} (W(t, p) - W(p, t)).\vec{\sigma}(t)$$

2. Some place is unmarkable, but then $\text{Reach}(P', T') = \emptyset$.

The second case is trivially semilinear; the first is the set of natural number solutions of a finite system of linear equations, which is an effectively semilinear set. $\square$

# Chapter 3

# Constructive Dickson's Lemma

## 3.1 Domination

*Domination* is a generalisation of the greater-than relation to monoids (§2.3):

$$f \geq g \quad =_{\text{def}} \quad \exists h. f = g + h \tag{3.1}$$

A sequence $f_1, f_2, \ldots$ is *domination-free* or *non-dominating* when no element dominates a predecessor: $\nexists i < j. f_i \leq f_j$. Clearly, on $(\mathbb{N}, +)$ (the natural numbers with addition), the length of every domination-free sequence $n_1, n_2, \ldots$ is both finite and bounded by $n_1$. Conversely, on $(\mathbb{N}, \times)$ there exists an infinite domination-free sequence: the prime numbers.

If we move to $(\mathbb{N}^2, +)$ (the two dimensional vector space with pointwise addition), it is no longer possible to put a finite bound on the length of a domination free sequence as a function of its first element. For example, from $(0, 1)$ we could have,

$$\begin{array}{ll} & (0,1),(0,0) \\ \text{or,} & (0,1),(1,0),(0,0) \\ \text{or,} & (0,1),(2,0),(1,0),(0,0) \\ \text{or,} & (0,1),(3,0),(2,0),(1,0),(0,0) \\ \ldots & \end{array}$$

all domination-free sequences. However, they *are* bounded in length by the second component of their second elements: no infinite domination-free sequence rooted at $(0, 1)$ exists. Dickson's Lemma states that in fact any domination-free sequence from $(\mathbb{N}^k, +)$ is finite.

**Lemma 3.1.1 (Dickson's Lemma, [Dic13])** *For any infinite sequence drawn from $\mathbb{N}^k$, $\vec{x_1}, \vec{x_2}, \ldots$ we can always find indices $i < j$ such that $\vec{x_i} \leq \vec{x_j}$.*

**Proof:** For a contradiction, let $\vec{x_1}, \vec{x_2}, \ldots \in \mathbb{N}^k$ be an infinite domination-free sequence. $\vec{x} \not\leq \vec{y} \implies \exists 0 < l \leq k.\vec{x}(l) > \vec{y}(l)$; that is, for each $j > 1$ there exists an index $0 < l \leq k$ such that,

$$\vec{x_1}(l) > \vec{x_j}(l) \tag{3.2}$$

The pigeonhole principle implies the existence of an infinite subsequence $y_1, y_2, \ldots$ with $\forall i.\vec{y_i}(l) < \vec{x_1}(l)$; a further application of the principle gives us an infinite subsequence $\vec{z_1}, \vec{z_2}, \ldots$ in which $\vec{z_1}(l) = \vec{z_2}(l) = \ldots$. Reapply the construction $k - 1$ times to yield an infinite sequence $\vec{v_1}, \vec{v_2}, \ldots$ in which $\forall i, j.\vec{v_i} = \vec{v_j}$: a *maximally* dominating sequence, yet any subsequence of a domination-free sequence must itself be domination free. □

It does not, however, help in putting ordinal bounds (§2.1) on the lengths of our non-dominating sequences. It is possible to say, for example, that the unfolding of a domination-free sequence of $\mathbb{N}^2$ is curbed in the following way: it is allowed a *finite* number of arbitrary increases in length, therefore it is limited strictly above by $\omega^2$. Before we prove this, a diversion into a forest.

### 3.1.1 Domination-free trees

A tree labelled from $\mathbb{N}^k$ is *domination-free* when every path through it forms a non-dominating sequence.

**Definition 3.1.1 (largest domination-free tree)** *The* maximal domination-free tree rooted at *$\vec{x} \in \mathbb{N}^k$, $DOM(\vec{x})$, is a tree $t$, with*

1. *The root of $t$ is labelled $\vec{x}$;*

2. *If $u$ is a node of $t$ and $\sigma = \vec{x_1}, \vec{x_2}, \ldots, \vec{x_m}$ is the sequence of labels in the path from $t$ to $u$, then for each $\vec{x'} \in \mathbb{N}^k$ such that $\sigma\vec{x'}$ is a domination-free sequence, add $u \rightarrow u'$ and label $u'$ by $\vec{x'}$.*

*I.e., $t$ holds all possible non-dominating sequences beginning with $\vec{x}$.* ■

Lemma 3.1.1 is equivalent to asserting that for any $\vec{x} \in \mathbb{N}^k$, $DOM(\vec{x})$ is well-founded. Returning to the sequences rooted at $(0, 1)$, we find,

$$h(DOM((0, 1))) = \omega \tag{3.3}$$

and for sequences of natural numbers, a *forest*: $DOM(0), DOM(1), DOM(2), \ldots$, whose height is $\sup\{h(DOM(n)) \mid n \in \mathbb{N}\} = \omega$ – i.e. the order type of domination-free sequences of natural numbers is $\omega$. We wish to find a full generalisation of this result, and thus a *constructive* version of Dickson's Lemma.

## 3.2 Lexicographically ordered sequences

A lower bound for the height of the maximal domination-free tree is provided by considering *lexicographically-ordered sequences*,

**Definition 3.2.1 (lexicographic ordering)** $\vec{x} >_{lex} \vec{y} =_{def} \exists i. \vec{y}(i) < \vec{x}(i) \wedge \forall j < i. \vec{y}(j) = \vec{x}(j)$ ■

In distinction to the non-domination relation $\not\leq$, $>_{\text{lex}}$ is a *total order*. For example, there are two permutations of $(0,1),(1,0)$ which make non-dominating sequences, $(0,1) \not\leq (1,0)$ and $(1,0) \not\leq (0,1)$, while for any $A \subseteq \mathbb{N}^k$ there is exactly one way of making a lexicographically ordered sequence of its elements, $\vec{x} >_{\text{lex}} \vec{y} \implies \vec{y} \not>_{\text{lex}} \vec{x}$[1]. (There are, in this sense, more non-dominating sequences of any given length than lexicographically-ordered ones.)

If $\sigma = \vec{x_1}, \vec{x_2}, \ldots$ is a lexicographically-ordered sequence from $\mathbb{N}^k$,

$$\vec{x_1} >_{\text{lex}} \vec{x_2} >_{\text{lex}} \cdots$$

it is easily seen that $\sigma$ is domination free ($\vec{x} >_{\text{lex}} \vec{y} \implies \exists l. \vec{y}(l) < \vec{x}(l) \iff \vec{x} \not\leq \vec{y}$). Define $LEX(\vec{x})$, the largest lexicographically-ordered tree rooted at $\vec{x}$ analogously to $DOM(\vec{x})$ (Definition 3.1.1). Clearly, $h(DOM(\vec{x})) \geq h(LEX(\vec{x}))$; we find the following (well known) result,

**Lemma 3.2.1 (The order-type of $>_{\text{lex}}$ on $\mathbb{N}^k$)** *For $\vec{x} \in \mathbb{N}^k$, $h(LEX(\vec{x})) < \omega^k$, and*

$$\sup\{h(LEX(\vec{x})) \mid \vec{x} \in \mathbb{N}^k\} = \omega^k$$

**Proof:** *Sketch.* An induction on $k$. Consider $\vec{x} = (a_1, \ldots, a_{n+1}) \in \mathbb{N}^{k+1}$. If the first component is pinned at $a_1$, we have (by hypothesis) a tree of height bounded strictly above by $\omega^k$. Each time the first component is reduced, we can raise the other components arbitrarily high while retaining the $>_{\text{lex}}$ order, that is, $a_1$ trees of height $\omega^k$, stacked upon each other. □

**Corollary 3.2.1** $\sup\{h(DOM(\vec{x})) \mid \vec{x} \in \mathbb{N}^k\} \geq \omega^k$

[1]I.e. it makes sense to write, $\not>_{\text{lex}} = <_{\text{lex}}$, but not to have $\not\leq = >$

## 3.3 A reification

We will construct a function $r : (\mathbb{N}^k)^+ \to \mathbb{N}^k$ with the property that, if $\vec{x_1}, \vec{x_2}, \ldots, \vec{x_n}, \vec{x_{n+1}}$ is non-dominating,

$$r(\vec{x_1}, \vec{x_2}, \ldots, \vec{x_n}) >_{\mathsf{lex}} r(\vec{x_1}, \vec{x_2}, \ldots, \vec{x_n}, \vec{x_{n+1}}) \tag{3.4}$$

This proof is taken from [HMS06], and owes to Faron Moller. For sequences $\sigma n \in \mathbb{N}^+$ we set $r(\sigma n) =_{\mathsf{def}} n$. For sequences of pairs, define, on $\sigma = \vec{x_1}, \vec{x_2}, \ldots, \vec{x_n} \in (\mathbb{N}^2)^+$,

$$\min_x(\sigma) =_{\mathsf{def}} \min\{\vec{x_j}(1) \mid 0 < j \leq m\} \quad \min_y(\sigma) =_{\mathsf{def}} \min\{\vec{x_j}(2) \mid 0 < j \leq m\} \tag{3.5}$$

and, second, the set of possible elements with which to extend a sequence while maintaining non-domination *and* its minimum values:

$$S_2(\sigma) =_{\mathsf{def}} \begin{array}{c}\{\vec{x} \mid \min_x(\sigma) \leq \vec{x}(1), \min_y(\sigma) \leq \vec{x}(2) \text{ and} \\ \forall 0 < i \leq m. \vec{x_i} \not\leq \vec{x}\end{array} \tag{3.6}$$

First, note that $S_2(\sigma)$ is always finite; second, that on $\sigma \in (\mathbb{N}^2)^+$, the following meets our needs:

$$r(\sigma) =_{\mathsf{def}} (\min_x(\sigma) + \min_y(\sigma), |S_2(\sigma)|) \tag{3.7}$$

That is, if $\sigma\vec{x} \in (\mathbb{N}^2)^+$ is domination-free, either:

1. $\min_x(\vec{x}) < \min_x(\sigma)$;
2. $\min_y(\vec{x}) < \min_y(\sigma)$; or
3. $\vec{x} \in S_2(\sigma)$ (i.e. $|S_2(\sigma)| \geq |S_2(\sigma\vec{x}) + 1|$)

hence, $r(\sigma) >_{\mathsf{lex}} r(\sigma\vec{x})$. We are now in a position to prove the result stated at the end of §3.1. Consider how a sequence of pairs rooted at $\vec{x}$ can unfold. $r(\vec{x}) = (x, y)$; in the next step, either we reduce the $y$ component (and keep $x$ constant), or reduce $x$ (and *raise* $y$ to whatever we please). That is, we have $x$ many opportunities to increase the potential length of our sequence arbitrarily far.

The construction proceeds by induction. Define a function to project out the $i$th component of a sequence,

$$\pi_{\text{-}i}((a_1, \ldots, a_n)) =_{\mathsf{def}} (a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n) \tag{3.8}$$

then, for $\sigma = \vec{x_1}, \vec{x_2}, \ldots, \vec{x_m} \in (\mathbb{N}^n)^+$, define the set of non-dominating subsequences of $\sigma$ in which the $i$th component has been deleted:

$$\begin{aligned} \mathrm{ND}_i(\sigma) \;&=_{\mathrm{def}}\; \{(\pi_{\text{-}i}(\vec{x_{i_1}}), \ldots, \pi_{\text{-}i}(\vec{x_{i_p}})) \,|\, p > 0, 0 < i_1 < \ldots < i_p \leq m \text{ and} \\ &\qquad (\pi_{\text{-}i}(\vec{x_{i_1}}), \ldots, \pi_{\text{-}i}(\vec{x_{i_p}})) \text{ is non-dominating}\} \end{aligned}$$

Finally,

$$\begin{aligned} \min_i(\sigma) \;&=_{\mathrm{def}}\; \min_{<_{\mathrm{lex}}}\{r_{n-1}(\sigma') \,|\, \sigma' \in \mathrm{ND}_i(\sigma)\} \\ S_n(\sigma) \;&=_{\mathrm{def}}\; \{\vec{x} \,|\, \forall 0 < i \leq n.\, \min_i(\vec{x}) = \min_i(\sigma\vec{x}) \text{ and} \\ &\qquad \forall 0 < j \leq m. \vec{x_j} \not\leq \vec{x}\} \end{aligned}$$

**Lemma 3.3.1** ($S_n$) *For $\sigma \in (\mathbb{N}^n)^+ . S_n(\sigma)$ is finite.*

**Proof:** For $\sigma = \vec{x_1}, \vec{x_2}, \ldots, \vec{x_m}$, we wish to prove that for any $\vec{x} \in S_n(\sigma)$, and any $0 < i \leq n$, there exists a $0 < j \leq m$ with $\vec{x}(i) < \vec{x_j}(i)$.

Let $0 < i \leq n$ and $i_1 < \ldots < i_p$ be such that,

$$\min_i(\sigma) = r_{n-1}(\pi_{\text{-}i}(\vec{x}_{i_1}), \ldots, \pi_{\text{-}i}(\vec{x}_{i_p}))$$

and suppose that $\vec{x} \in S_n(\sigma)$. If $(\pi_{\text{-}i}(\vec{x_{i_1}}), \ldots, \pi_{\text{-}i}(\vec{x_{i_p}}))$ is non-dominating, then by induction,

$$\begin{aligned} \min_i(\sigma\vec{x}) \;&\leq_{\mathrm{lex}}\; r_{n-1}(\pi_{\text{-}i}(\vec{x_{i_1}}), \ldots, \pi_{\text{-}i}(\vec{x_{i_p}}), \pi_{\text{-}i}(\vec{x})) \\ &<_{\mathrm{lex}}\; r_{n-1}(\pi_{\text{-}i}(\vec{x_{i_1}}), \ldots, \pi_{\text{-}i}(\vec{x_{i_p}})) \\ &=\; \min_i(\sigma) \end{aligned}$$

But then, $\vec{x} \notin S_n(\sigma)$. $\pi_{\text{-}i}(\vec{x_{i_1}}), \ldots, \pi_{\text{-}i}(\vec{x_{i_p}}), \pi_{\text{-}i}(\vec{x})$ is dominating: $\exists j. \pi_{\text{-}i}(\vec{x_{i_j}}) \leq \pi_{\text{-}i}(\vec{x})$, yet $\vec{x_{i_j}} \not\leq \vec{x}$, hence $\vec{x}(i) < \vec{x_{i_j}}(i)$. Since we can do this for each component of $\vec{x}$, the members of $S_n(\sigma)$ must be bounded as a function of $\sigma$, and be finitely many. □

Finally, define

**Definition 3.3.1** ($r_n$) *For $\sigma \in (\mathbb{N}^n)^+$,*

$$r_n(\sigma) =_{def} \left( \sum_{i=1}^{n} \min_i(\sigma), |S_n(\sigma)| \right)$$

*where the sum is componentwise (i.e. $r_n(\sigma) \in \mathbb{N}^n$).* ■

**Lemma 3.3.2** $(r_n)$ *For $\sigma \in (\mathbb{N}^n)^+$, $\vec{x} \in \mathbb{N}^n$, if $\sigma\vec{x}$ is non-dominating then,*

$$r_n(\sigma) >_{lex} r_n(\sigma\vec{x})$$

**Proof:** For all $i$, $\mathrm{ND}_{-i}(\sigma) \subseteq \mathrm{ND}_{-i}(\sigma\vec{x})$, hence $\min_i(\sigma\vec{x}) \leq \min_i(\sigma)$. If equality holds in all cases, then $S_n(\sigma\vec{x}) \subsetneq S_n(\sigma)$, since $S_n(\sigma\vec{x}) \subseteq S_n(\sigma)$, but $\vec{x} \in S_n(\sigma)$. That is to say, $r_n(\sigma\vec{x}) <_{\text{lex}} r_n(\sigma)$. $\square$

We are left with the situation that, although non-domination is a strictly weaker condition than lexicographical ordering, nevertheless:

**Theorem 3.3.1 (non-dominating vector-labelled trees)**
*If $t$ is a non-dominating tree labelled from $\mathbb{N}^k$, then $h(t) < \omega^k$.*

**Proof:** Label each node $u$ of $t$ by $r_n(\sigma)$, where $t \to t_1 \to \ldots \to t_m \to u$ is the path leading to $u$, and $\sigma = l(t), l(t_1), \ldots, l(t_m), l(u)$ is the (necessarily domination-free) sequence of labels. By Lemma 3.3.2 we have a lexicographically-ordered tree, and applying Lemma 3.2.1 gives us $h(t) < \omega^k$. $\square$

## 3.4 Further and related work

(This work forms part of [HMS06], and first appeared as a CALCO-jnr 2005 Swansea University Research Report in [HM05]; it was presented at CSL 2006 by Anton Setzer.) Other constructive proofs of Dickson's Lemma have been undertaken, in particular in the field of term rewriting. Martín-Mateos *et al* formalised the proof of Dickson's Lemma using the ACL2 theorem prover[2] in [MMAHRR03] (without an ordinal mapping), while at the same time and for the same end Sustik produced an explicit ordinal mapping on sequences[3], [Sus03], though not an optimal one, giving only an $\epsilon_0$ result – so, for example, the bound on *pairs* is already $\omega^\omega$.

As noted by Setzer, very recently – preprint March 2006, [BG06] – Blass and Gurevich have defined the *stature*, $||P||$, of a well partial ordering $P$ as the order type of nondominating sequences of $P$; they derive the $\omega^k$ result found in Theorem 3.3.1 as a special case. Their interest is in program termination (using ordinals is an approach that goes back at least as far as Turing, [MJ84]); their proofs are both more general and more difficult than ours.

[2] `http://www.cs.utexas.edu/users/moore/acl2/`

[3] The stated motivation behind such work has been the formalisation of Buchberger's Algorithm, an important tool for creating Gröbner bases. Note that Stříbrná has found a nice connection between Basic Parallel Process bisimilarity and polynomial rings, which itself uses Gröbner bases for decidability, [Stř99].

# Chapter 4

# General Processes

A process *acts*, and becomes a new process.

## 4.1 Labelled Transition Systems

A unifying view for the automata and process algebras glossed in the Introduction, which stems from the *Structural Operational Semantics* of Plotkin [Plo81, AFV01], is the *labelled transition system.*

**Definition 4.1.1 (LTS)**
*A* Labelled Transition System *is a directed labelled graph: a tuple* $(S, \Sigma, \rightarrow)$ *of* states $S$, action names $\Sigma$ *and* transition relation $\rightarrow \subseteq S \times \Sigma \times S$. ■

At this level of description no weights are given to the actions, no propensities, stochastic or otherwise, no timing[1], no distinction between causal and contingent, and no true concurrency. The processes that can be described by LTSs we name the *general processes.*

**Definition 4.1.2 (general processes)** *A* general process *is a pair* $(L, u)$, *where* $L = (S, \Sigma, \rightarrow)$ *is a LTS, and* $u \in S$. *That is, a rooted LTS.* ■

If $(L, u)$ is a general process, and $u \xrightarrow{a} v$ is a transition in $L$, we say (returning to this chapter's opening) that $u$ may perform the action $a$ to become the process $v$. We will typically refer to a process $(L, u)$ as $u$, its LTS left implicit. The collection of all general processes is a proper class (since there is no set of graphs); everything that follows will take place within it.

We will later garnish the $\xrightarrow{a}$ relation with information as to the change wrought by the action modulo some function (§6.1, Equation §6.5), and the (ordinal) number

[1]See e.g. [Emm88].

of times this can be done (§6.4). But for now, a standard notation for when an action cannot be made,

$$u \overset{a}{\not\rightarrow} \quad \text{:iff} \quad \nexists u'.u \xrightarrow{a} u' \tag{4.1}$$

and the usual extension from actions to sequences of actions: $\forall u.u \xrightarrow{\epsilon} u$, and if $w = (a_1, \ldots, a_{n+1}) \in \Sigma^+$,

$$u \xrightarrow{w} u' \quad \text{:iff} \quad \exists u_1, u_2, \ldots, u_n.u \xrightarrow{a_1} u_1 \ldots \xrightarrow{a_n} u_n \xrightarrow{a_{n+1}} u' \tag{4.2}$$

Moreover, $u \xrightarrow{w^*} u'$ :iff $\exists n.u \xrightarrow{w^n} u'$.

## 4.1.1 Simple classes of general processes

A *void* process is one whose only transition is the *void transition*, $u \xrightarrow{\epsilon} u$,

$$\text{void}(u) \quad \text{:iff} \quad \forall a \in \Sigma.u \overset{a}{\not\rightarrow} \tag{4.3}$$

The *norm* of a process is the minimal distance from it to a void process,

$$\text{norm}(u) = \min\{|w| \mid u \xrightarrow{w} u' \wedge \text{void}(u')\} \tag{4.4}$$

(By convention, $\min \emptyset = \infty$.) A LTS $L = (S, \Sigma, \rightarrow)$ is *normed* when all $u \in S$ have finite norm. A *process* is normed when it has a finite norm, and cannot reach a process with infinite norm. Our formal definition is a little stronger:

**Definition 4.1.3 (normed processes)**
*$(L, u)$ is a* normed process *when $L$ is normed.* ■

A LTS is *well-founded* (§2.1.2) when every sequence of transitions $u \xrightarrow{a_1} u_1 \xrightarrow{a_2} u_2 \ldots$ within it is finite (a sufficient but not a necessary condition for normedness, Figure 4.1).

$$\begin{tikzcd} u \arrow[loop left, "a"] \arrow[r, "a"] & {} \end{tikzcd}$$

Figure 4.1: A normed but non-well-founded LTS

**Definition 4.1.4 (well-founded processes)** *$(L, u)$ is a* well-founded process *when no infinite paths exist through $L$.* ■

The well-founded processes are a class intimately bound to the ordinal numbers (§2.1). Define the *height* of a process,

$$h(u) =_{\text{def}} \sup\{h(v) + 1 \mid u \xrightarrow{a} v\} \tag{4.5}$$

then, $h(u) \in \mathcal{O}$ if and only if $u$ is a well-founded process, and moreover for every ordinal $\kappa \in \mathcal{O}$ there exists a well-founded process $u$ with $h(u) = \kappa$ (see Lemma 2.1.2).

A transition $u \xrightarrow{a} u'$ is *norm reducing* when $\infty > \text{norm}(u) > \text{norm}(u')$ ($\text{norm}(u) = \text{norm}(u') + 1$). A LTS is normed if and only if every $u \in S$ possesses at least one norm-reducing transition; the LTS is well-founded if (but not only-if) all transitions are norm reducing.

If, for each $u \in S(L)$, there is at most one transition of any given label (i.e. $u \xrightarrow{a} u'$ and $u \xrightarrow{a} u''$ implies $u' = u''$) then $L$ is termed *deterministic*. If in the LTS of a process $u$ there can only ever be finitely many choices for the next state (as is true of all of the systems introduced in the previous chapter) we say that $u$ is finitely branching, or, *sub-$\aleph_0$-branching* (see §2.2).

Generalising,

**Definition 4.1.5 (sub-$\aleph$-branching processes)**

*$(L, u)$ is* sub-$\aleph$-branching *when the vertex degree of $L$ is bounded strictly above by $\aleph$.* ■

Further, the sub-$\aleph_1$-processes correspond to those whose branching is countable. A studied subset are those processes which can be represented as vectors from $\mathbb{N}^k$. For example, Petri Nets (§2.5.2) and BPPs (Chapter 5), with or without *silent actions* (§4.5).

**Definition 4.1.6 ($k$-tuple processes)**

*$(L, u)$ is a $k$-tuple process when $S(L) \simeq \mathbb{N}^k$.* ■

## 4.2 Equivalence relations on processes

Given two processes, the first question one might ask is whether they represent the same system: are they equivalent? There are dozens of studied notions of equivalence (many summarised by van Glabbeek in [vG90]). Since processes are, for our purposes, directed labelled graphs, a natural first attempt is *graph equivalence*: that they share an identical structure. But a process essentially models *action*, and it proves easy to find an example of structurally different graphs whose behaviours are indistinguishable. Our second attempt will draw from automata theory: processes are equivalent when they can exhibit identical sequences or traces of actions. But here we will find the notion too permissive – necessary but not sufficient. This holds too of a variant known as *failures equivalence*, the native idea of equivalence for Hoare's CSP [Hoa78a]. The concept we will reach, *bisimilarity*, owes to David Park [Par81] (though it has antecedents in logic: p-morphisms [Seg71], and zig-zag relations [vB76]), and was originally used in Milner's CCS [Mil80, Mil89].

We draw a line between a *static* or *timeless* notion of equivalence and an *operational* one; and subdivide the latter into *linear* and *branching time*[2]. Our aim is to find a definition of equivalence which captures reactive behaviour.

Static notions of equivalence treat processes as objects, with no idea that they *do* anything – generate a language, model a system.

**Definition 4.2.1 (Graph equivalence)** *$(L, u)$ and $(M, v)$ are* graph equivalent *$u \simeq v$ iff there exists an isomorphism $f : L \to M$ and $f(u) = v$.* ∎

This is an equivalence relation, yet one too restrictive when we consider processes in terms of what they can do (an idea which will become more precise in the course of this chapter). $u$ and $v$ of Figure 4.2 are not structurally equivalent, but act identically; since a *process* is about action, we would not wish to distinguish them, nor use a notion of equivalence which does.

$$\begin{tikzcd} u \arrow[loop right, "a"] & & v \arrow[r, bend left, "a"] & t \arrow[l, bend left, "a"] \end{tikzcd}$$

Figure 4.2: $u \not\simeq v$

### 4.2.1 Trace and failures equivalence

A *trace* of a process $u$ is a sequence of actions $w \in \Sigma^*$ with $u \xrightarrow{w}$ (see Equation 4.2, page 41). A *completed trace* from $u$ is a sequence of actions which brings $u$ to a void process, $u \xrightarrow{w} v \wedge \text{void}(v)$. Two processes are (completed) trace equivalent when their sets of (completed) traces are equal:

**Definition 4.2.2 (trace equivalence)**

$$L(u) =_{def} \{w \mid u \xrightarrow{w} v \wedge void(v)\} \quad LT(u) =_{def} \{w \mid u \xrightarrow{w}\} \tag{4.6}$$

*Two processes $u, v$ are* trace equivalent, *$u \equiv_{LT} v$ when $LT(u) = LT(v)$. They are* completed trace equivalent *or* language equivalent, *$u \equiv_L v$, when $L(u) = L(v)$.* ∎

(See the introduction to Chapter 5 for more on completed traces and *languages*.) Considering again the processes of Figure 4.2, $LT(u) = \{a^n \mid n \in \mathbb{N}\} = LT(v)$ and $L(u) = L(v) = \emptyset$. However, completed trace equivalence only makes sense for normed

[2]Likewise, temporal logics are commonly characterised as either linear (e.g. Linear Temporal Logic), each moment has only one future, or branching, taking into account a multiplicity of future worlds (e.g. Computation Tree Logic) – see [Var01, HR04].

processes (4.1.1), while trace equivalence fails to capture *deadlock*[3]. The processes $u, v$ of Figure 4.3 have the same traces, but while one can halt, the other cannot.

$$\begin{tikzcd}
u \arrow[loop above, "a"] & v \arrow[l, "a"'] \arrow[r, "a"'] & {}
\end{tikzcd}$$

Figure 4.3: $v$ is capable of deadlocking, $u$ is not

To account for deadlock, we might follow CSP [Hoa78a, BHR84] and generalise completed traces to *failures equivalence*:

**Definition 4.2.3 (failures equivalence)** *A* failure *of $u$ is a pair $(w, A)$, with $w \in \Sigma^*, A \subseteq \Sigma$, and $\exists u'.u \xrightarrow{w} u' \wedge \forall a \in A.u' \overset{a}{\not\rightarrow}$. $f(u)$ denotes the set of failures of $u$; two processes are* failures equivalent, *$u \equiv_f v$, when they have the same failures.* ■

Traces and failures are *linear-time* notions of equivalence: they take no notice of the branching structure of processes. This structure is, however, important if we wish to truly capture reactive behaviour. In Figure 4.4, $L(s) = L(t) = \{ab, ac\}$, but after the initial $a$ action $s$ evolves into a process capable of doing $b$ or $c$, while $t$ has to choose, either moving to a state which can perform $b$, or to one with a $c$ action enabled. The processes of Figure 4.5 have the same failures sets, but their behaviour clearly differs. *Branching time* semantics pulls us back towards isomorphism: behaviour cannot be determined as a set of runs, as this ignores divergent action [vG94]. Its most studied example is *bisimilarity.*

$$\begin{tikzcd}
 & s \arrow[d, "a"] & t \arrow[d, "a"] \arrow[dr, "a"] & \\
 & s_1 \arrow[dl, "b"'] \arrow[d, "c"] & t_1 \arrow[d, "b"] & t_2 \arrow[d, "c"] \\
{} & {} & {} & {}
\end{tikzcd}$$

Figure 4.4: $s \equiv_L t$

## 4.3 Bisimulation equivalence

In [Ace03], Luca Aceto gives the following as a prominent open question,

> Can one prove in a formal sense that bisimulation equivalence is the finest reasonable behavioural equivalence?

[3]For our purposes, a process is deadlocked iff it is void.

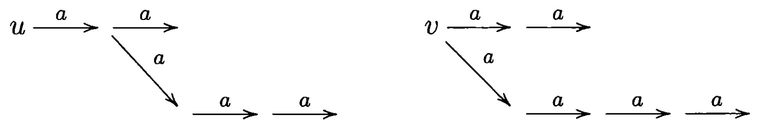


Figure 4.5: $u \equiv_f v$

Intuitively, two processes are *bisimilar* when whatever one can do, the other can match, such that the resulting processes are still bisimilar. Technically, processes are bisimilar when they are related by a *bisimulation*:

**Definition 4.3.1 (Bisimulation)** *A binary relation $R$ on general processes is a* bisimulation relation *exactly when $uRv$ implies,*

1. *If $u \xrightarrow{a} u'$ then $\exists v'.v \xrightarrow{a} v'$ s.t. $u'Rv'$; and*

2. *If $v \xrightarrow{a} v'$ then $\exists u'.u \xrightarrow{a} u'$ s.t. $u'Rv'$*

*Two processes $u, v$ are* bisimilar

$$u \sim v$$

*if and only if they are related by a bisimulation; that is, $\sim$ is the union of all bisimulation relations.*[4] *(Note that we have elided the labelled transition parts of the general processes in our definition.)* ■

**Example 4.1 (bisimilar and non-bisimilar processes)** *The processes of Figure 4.2 are bisimilar, while those of Figures 4.4 and 4.5 are not. For the former, imagine we have a bisimulation $R$ with $sRt$. Then, since $t \xrightarrow{a} t_1$, we must have $s_1 R t_1$, but $s_1 \xrightarrow{c}$ and $t_1 \overset{c}{\not\rightarrow}$.*

**Lemma 4.3.1 (bisimilarity on deterministic processes)** *On deterministic processes, $\sim = \equiv_{LT}$; on normed deterministic processes, $\sim = \equiv_L$.*

**Proof:** Since $\exists w.u \xrightarrow{w} \iff v \overset{w}{\not\rightarrow} \implies u \not\sim v$ it suffices to prove that $\equiv_{LT}$ and $\equiv_L$ are bisimulation relations on deterministic and normed deterministic processes respectively.

For the former, let $u \equiv_{LT} v$, and wlog $u \xrightarrow{a} u'$. Since there is exactly one action $v \xrightarrow{a} v'$ from $v$, and the sets of $a$-prefixed words of $u, v$ are identical, it must be that

[4]A *simulation* relation is defined by using only the first clause; $u$ simulates $v$ iff they are related by a simulation relation. Note that mutual simulation does not imply bisimilarity (easily seen), and in general simulation is a harder problem to decide – for example, bisimilarity is decidable on BPA §5.3, simulation is undecidable already for normed BPA, [GH94]. In [KM02c], Kučera and Mayr show that there exists a polynomial reduction of bisimilarity to simulation equivalence across a wide range of processes.

$u' \equiv_{LT} v'$. The sequel is similar; we only need note that *normedness* implies that any action $u \xrightarrow{a} u'$ contributes towards some completed trace. (For a discussion of bisimilarity and language equivalence, see [Sti01a].) $\square$

### 4.3.1 Bisimilarity on finite processes

$(L, u)$ is a *finite process* when $L$ is a finite graph. The language equivalence problem is known to be PSPACE-complete on such processes [MS72], whereas bisimilarity is P-complete [ÁBGS91], and indeed:

**Lemma 4.3.2** ($\sim$ **on finite processes)**
*Bisimilarity is decidable on finite processes in $O(n \log n)$ time.*

**Proof:** [Mol00]. See [KS90, PT87] for the *bisimulation colouring* algorithm. $\square$

### 4.3.2 Bisimulation games

Bisimilarity admits a natural game-theoretic characterisation,

**Definition 4.3.2** ($G(u, v)$) *The game $G(u_1, u_2)$ is played between I and II. First, I chooses a process $u_i$ and a transition (which may include the void transition, $u_i \xrightarrow{\epsilon} u_i$),*

$$I : u_i \xrightarrow{a} u_i' \tag{4.7}$$

*to which II must respond with a transition of the same action-name from the other process,*

$$II : u_{3-i} \xrightarrow{a} u_{3-i}' \tag{4.8}$$

*If no such move exists, the game ends. Otherwise, continue with $G(u_1', u_2')$. All finite games are won by I; all infinite games, II.*[5] $\blacksquare$

The following is a standard result (see e.g. [Sti01d]):

**Theorem 4.3.1** ($G(u, v)$) *$u \sim v$ iff II has a winning strategy for $G(u, v)$.*

**Example 4.2** ($\sim \subsetneq \equiv_{LT}$) *To prove that $\sim \subsetneq \equiv_{LT}$ it suffices to show that* if *there exists a $w$ with $u \xrightarrow{w} \iff v \overset{w}{\not\rightarrow}$* then *I has a winning strategy on $G(u, v)$. (That strategy being, if wlog $u \xrightarrow{w} \wedge v \overset{w}{\not\rightarrow}$, at each turn, successively perform $u \xrightarrow{a_1} u_1 \xrightarrow{a_2} u_2 \xrightarrow{a_3} \ldots \xrightarrow{a_n} u'$, where $w = a_1 \ldots a_n$; there must come a transition against which II has no response.)*

[5]Note, the more common definition of the game does not allow $u \xrightarrow{\epsilon} u$ transitions, and adds to the winning conditions that the first player who cannot make a move loses. These definitions are equivalent; the one used here has the advantage that it is identical (with $\overset{a}{\Rightarrow}$ for $\xrightarrow{a}$) to the *weak* bisimulation game, and simplifies the construction of *optimal move trees*, §4.4.1.

### 4.3.3 Bisimulation approximants

Rather than require that the game must be infinite for $II$ to win, we set an ordinal bound, moving from $G$ to $G_\kappa$. The game $G_\kappa(u_1, u_2)$ is played between $I$ and $II$. If $\kappa = 0$, player $II$ wins. Otherwise, $I$ chooses a process $u_i$, one of its transitions $u_i \xrightarrow{a} u_i'$ and a $\mu < \kappa$; $II$ has to respond with an identically-named move from $u_{3-i}$. If there are none, $I$ wins; otherwise the game proceeds on $G_\mu(u_1', u_2')$[6].

Turning from games to relations, the bisimulation approximants, or stratified bisimulations, are defined:

**Definition 4.3.3 (Bisimulation approximants)** *The* bisimulation approximants $\sim_\kappa$, *for all ordinals* $\kappa \in \mathcal{O}$, *are*

*1.* $u \sim_0 v$ *for all processes* $u$ *and* $v$.

*2.* $u \sim_{\kappa+1} v$ *iff*

*(a) if* $u \xrightarrow{a} u'$ *there is a transition* $v \xrightarrow{a} v'$ *such that* $u' \sim_\kappa v'$*;* *and*

*(b) if* $v \xrightarrow{a} v'$ *there is a transition* $u \xrightarrow{a} u''$ *such that* $u' \sim_\kappa v'$.

*3. For all limit ordinals* $\lambda$, $u \sim_\lambda v$ *iff* $u \sim_\kappa v$ *for all* $\kappa < \lambda$.

■

**Lemma 4.3.3** *$II$ has a winning strategy for $G_\kappa(u, v)$ iff $u \sim_\kappa v$.*

For convenience, $\alpha \sim^!_\kappa \beta$ abbreviates $\alpha \sim_\kappa \beta \wedge \alpha \not\sim_{\kappa+1} \beta$.

**Example 4.3** *In Figure 4.6, $II$ has a winning strategy on $G_\omega(u, v)$, but not for $G_{\omega+1}(u, v)$, therefore $u \sim^!_\omega v$.*

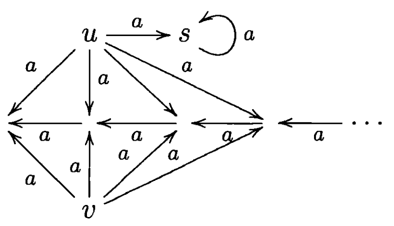


Figure 4.6: The Fan, $u \sim^!_\omega v$

[6]Implicitly, $G = G_\infty$, where $I$ never winning is equivalent to a victory for $II$.

**Example 4.4 (trace equivalence and finite approximability)**

$\sim_\omega \subsetneq \equiv_L$. *Imagine that* $u \not\equiv_L v$, *i.e.* $\exists w. u \xrightarrow{w} \iff v \overset{w}{\not\rightarrow}$. *Clearly I has a winning strategy on* $G_{|w|}(u,v)$*: perform, one by one, the sequence of actions w from the process which is capable of doing it. To make the inclusion strict, note that (in Figure 4.6),* $s \equiv_L t$*, but* $s \not\sim_2 v$.

**Example 4.5 (failures and readies equivalences)** *See Definition 4.2.3. The* readies *of a process are [OH86],*

$$r(u) \quad = \quad \{(w,A)\,|\,\exists u'.u \xrightarrow{w} u'.\forall a \in A.u' \xrightarrow{a} \iff a \in A\}$$

$f(u) \neq f(v)$ *or* $r(u) \neq r(v)$ *gives I a winning strategy on the game* $G_\omega(u,v)$. *Let* $(w,A)$ *witness inequivalence, then I can force play to a pair of processes* $u', v'$ *such that* $\exists a \in A.u' \xrightarrow{a} \iff v' \overset{a}{\not\rightarrow}$.

## 4.3.4 Semideciding $\not\sim$ on finitely-branching processes

On finitely branching processes with an effective next-state relation (a description which includes Petri Nets and pushdown automata, §2.5) we are able to semidecide non-bisimilarity by playing the bisimulation game on successively larger finite graphs.

If $u, v$ are processes of finitely branching LTSs, the set of reachable states in the game $G_n(u,v)$ is itself finite[7]. If the next-state relation is effective, this finite graph is computable. The game $G_n(u,v)$ is then equivalent to $\sim$ on finite processes (*any* $u'$, $v'$ more than $n$ transitions away from $u$, $v$ respectively can be removed without affecting the outcome of $G_n(u,v)$) – which is decidable (Lemma 4.3.2). By Lemma 4.4.4, $u \not\sim v \implies u \not\sim_\omega v \implies \exists n.u \not\sim_n v$; we conclude that:

**Lemma 4.3.4 (semideciding $\not\sim$ on finitely-branching processes)**

$\not\sim$ *is semidecidable on finitely-branching processes with an effective next-state relation.*

(It is easy to define a process which branches finitely but cannot have an effective next-state relation. Name the root $\lambda$. The children of a node $u$ are $u0$ and $u1$, and if $u$ is represents the binary code (modulo some fixed universal Turing Machine) of a program $M$ which halts given its own Gödel number as input then $u$ has an additional transition, $u \xrightarrow{halts}$. This LTS cannot be explored.)

[7]Moreover, if their branching is bounded above by some $N$, the set of possible states for $G_n$ is at most $N^{2^n}$.

### 4.3.5 Finite approximants with finite alphabets

The zeroth approximant $\sim_0$ relates all processes; the first approximant $\sim_1$ relates those processes $u, v$ for which,

$$\forall a \in \Sigma . u \xrightarrow{a} \iff v \xrightarrow{a} \tag{4.9}$$

If $\Sigma$ is a finite alphabet, the total number of equivalence classes modulo $\sim_1$ is simply $2^{|\Sigma|}$. That is, given any set of processes $\mathcal{X}$,

$$|\mathcal{X} / \sim_1 | \leq 2^{|\Sigma|} \tag{4.10}$$

Imagine there are $N$ possible equivalence classes modulo $\sim_n$, $T_1, \ldots, T_N$, and $\Sigma = \{a_1, \ldots, a_m\}$. Considering $\sim_{n+1}$, a process is either able to perform an $a_1$ action to reach a process in $T_1$ or not; able to perform an $a_1$ to a process in $T_2$, or not, and so forth. (Note that we have assumed nothing about the branching structure of these processes.) We find,

**Lemma 4.3.5 (Finite approximants over finite alphabets)**

*Let* $| \sim_n | =_{def} \sup\{|\mathcal{A}/ \sim_n | \,|\, \mathcal{A}$ *a set of processes over the finite alphabet* $\Sigma\}$, *then*

$$| \sim_n | = 2^{\left.|\Sigma| 2^{|\Sigma| 2^{|\Sigma| \cdots ^{2^{|\Sigma|}}}}\right\}n} \tag{4.11}$$

We will use this result in Chapter 6 (in the proof of Lemma 6.2.3). Of interest for its own sake is the situation when $\Sigma$ is infinite. This is not an avenue we have explored, but the Generalised Continuum Hypothesis appears to imply that,

$$|\Sigma| = \aleph_i \implies | \sim_{n+1} | = \aleph_{i+n+1} \tag{4.12}$$

## 4.4 Approximant hierarchy

On general processes there exists a strict approximant hierarchy,

**Lemma 4.4.1 (approximant hierarchy)** $\mu < \kappa \implies \sim_\kappa \subsetneq \sim_\mu$

**Proof:** For any $\kappa in \mathcal{O}$ we wish to (transfinite inductively) construct two processes $u, v$ with $u \sim^!_\kappa v$. Define $u_o$ to be a void (actionless) process, void$(u_0)$. Given processes $u_\mu$ for all $\mu < \kappa$, let $u_\kappa$ have exactly the transitions, $\forall \mu < \kappa . u_\kappa \xrightarrow{a} u_\mu$. We will prove that,

$$\kappa < \kappa' \implies u_\kappa \sim^!_\kappa u_{\kappa'} \tag{4.13}$$

by an induction on $\kappa$. Of course, if $\kappa > 0$ then $u_0 \sim^!_0 u_\kappa$. Let $\kappa < \kappa'$ and consider the game $G_\kappa(u_\kappa, u_{\kappa'})$. To show that $u_\kappa \sim^!_\kappa u_{\kappa'}$, note that any move from $u_\kappa$ can be replicated exactly from $u_{\kappa'}$, so player $I$ must move $u_{\kappa'} \xrightarrow{a} u_{\kappa''}$ with $\kappa'' \geq \kappa$. But now, for any $\mu < \kappa$ $II$ can move $u_\kappa \xrightarrow{a} u_\mu$, and by hypothesis $u_\mu \sim^!_\mu u_{\kappa''}$ (but $II$ has no move $u_\kappa \xrightarrow{a} u_\kappa$). □

**Example 4.6 (approximant collapse on finitely branching processes)**
*Let $u \sim^!_\omega v$, and wlog $I : u \xrightarrow{a} u'$. By definition, for each $i < \omega$ we can find a process $v \xrightarrow{a} v_i$ such that $v_i \sim_i u'$ but $v_i \not\sim_\omega u'$. This is only possible if $v$ can reach infinitely many processes in a single step, hence on all finitely branching processes, $\sim = \sim_\omega$.*

### 4.4.1 Optimal move trees

Informally, on a game $G(u,v)$, $u \xrightarrow{a} u'$ is an *optimal move* for Player $I$ if no other move exists which will end the game sooner. Formally, if $u \sim^!_\kappa v$, $u \xrightarrow{a} u'$ is an *optimal move* for Player $I$ iff there does not exist a $v \xrightarrow{a} v'$ with $u' \sim_\kappa v'$. If $u \sim v$, all moves are optimal (i.e. equally suboptimal).

**Definition 4.4.1 (omt)** *$omt(u,v)$ is constructed,*

1. *The root is $(u,v)$.*
2. *For a node $(s,t)$,*
   (a) *If $s \xrightarrow{a} s'$ is an optimal move for $I$ then, for all $t'.t \xrightarrow{a} t'$, $(s,t) \rightarrow (s',t')$; and*
   (b) *If $t \xrightarrow{a} t'$ is an optimal move for $I$ then, for all $s'.s \xrightarrow{a} s'$, $(s,t) \rightarrow (s',t')$*

*(If $II$ has no responses, i.e. $u \not\sim_1 v$, the tree will be empty (denoted by $\emptyset$).)* ■

Then, $u \not\sim v$ if and only if $omt(u,v)$ is well-founded. And further,

**Lemma 4.4.2 (omt)** *$u \sim^!_\kappa v$ iff $h(omt(u,v)) = \kappa$*

**Proof:** $u \sim^!_0 v$ iff $\exists a.u \xrightarrow{a} \iff v \overset{a}{\not\rightarrow}$ iff $omt(u,v) = \emptyset$ iff $h(omt(u,v)) = 0$. Let $u \sim^!_\kappa v$, and $u \xrightarrow{a} u_1, u_2, \ldots$, $v \xrightarrow{a} v_1, v_2, \ldots$ be the optimal moves for $I$ (not necessarily a countable sequence). For each of $II$'s possible responses, $u \xrightarrow{a} u'$, $v \xrightarrow{a} v'$ (respectively) we have (by hypothesis) $h(omt(u_i,v')), h(omt(u',v_i)) < \kappa$, while for any $\mu < \kappa$ we can find a pair $(u_i,v')$ or $(v',u_i)$ with $h(omt(u_i,v')) > \mu$ or $h(omt(u',v_i)) > \mu$. That is, $h(omt(u,v)) = \sup\{\mu + 1 \mid \mu < \kappa\} = \kappa$. Conversely, $h(omt(u,v)) = \kappa$ implies the existence of a winning strategy for $II$ on $G_\kappa(u,v)$, and a winning strategy for $I$ on $G_{\kappa+1}(u,v)$. □

### 4.4.2 Approximant hierarchy collapse

**Lemma 4.4.3 (collapse on sets of processes)** *On any set $S$ of processes there exists a $\kappa$ s.t. $\sim = \sim_\kappa$.*

**Proof:** For each pair $(u,v) \in S \times S$, either $u \sim v$ or $\exists \mu . u \not\sim_\mu v$. Let $C = \{\mu \mid u \not\sim_\mu v\}$; either $C$ is a set, in which case $\kappa = \sup C$, or $C$ is a proper class, but then so must $S$ be. □

**Example 4.7 (well-founded processes)** *If $u, v$ are well-founded processes, and $\kappa = \max\{h(u), h(v)\}$, then $u \sim v$ iff $u \sim_\kappa v$ (an induction on $\kappa$).*

**Lemma 4.4.4 (sub-$\aleph$-branching approximant collapse)**
*On the sub-$\aleph$-branching processes, where $\aleph$ is a regular cardinal (§2.2.1),*

$$\sim = \sim_\aleph$$

**Proof:** Imagine there exist sub-$\aleph$-branching processes $u, v$, for a regular $\aleph$, with $u \sim^!_\kappa v$ for $\kappa \geq \aleph$. $omt(u,v)$ is a sub-$\aleph$-branching tree, hence $h(omt(u,v)) < \aleph$ (Theorem 2.2.1), but $h(omt(u,v)) = \kappa \geq \aleph$. □

In particular, it is well known that on finitely branching processes, $\sim = \sim_\omega$ (Example 4.6), and on countably-branching processes (e.g. the $k$-tuple processes, Definition 4.1.6),

$$\sim = \sim_{\omega_1} \tag{4.14}$$

## 4.5 Silent actions

We use a $\tau$ to denote a *silent action* (as opposed to an observable or *strong* action $a \neq \tau$)[8]. While a (strong) transition $u \xrightarrow{a} u'$ involves a single step through the LTS of $u$, a *weak* transition $u \overset{a}{\Rightarrow} u'$ ($a \in \Sigma \cup \{\epsilon\}$) incorporates any number of silent moves, and at most one strong action. Formally,

$$u \overset{a}{\Rightarrow} u' \quad \text{:iff} \quad \begin{cases} u \xrightarrow{\tau^*} u' & \text{if } a = \epsilon \\ u \xrightarrow{\tau^* a \tau^*} u' & \text{otherwise} \end{cases} \tag{4.15}$$

[8]In CCS-like formalisms – for instance, BPP$_\tau$ of §5.2 – $\tau$ denotes an (internal) synchronisation; in ACP$_\tau$, [JJ85], observable actions are *abstracted* into silent actions, renamed $\tau$.

Every concept introduced so far has a natural weak equivalent. Weak trace and completed trace equivalence, $\equiv_{\tau LT}, \equiv_{\tau L}$, become

$$u \equiv_{\tau LT} v \quad =_{\text{def}} \quad \forall w \in \Sigma^*. u \overset{w}{\Rightarrow} \iff v \overset{w}{\Rightarrow} \tag{4.16}$$

$$u \equiv_{\tau L} v \quad =_{\text{def}} \quad \forall w \in \Sigma^*. u \overset{w}{\Rightarrow} u' \wedge \text{void}(u') \tag{4.17}$$

$$\iff v \overset{w}{\Rightarrow} v' \wedge \text{void}(v') \tag{4.18}$$

while a *weak bisimulation relation* is,

**Definition 4.5.1 (weak bisimulation)** *$R$ is a* weak bisimulation relation *when $uRv$ implies, for $a \in \Sigma \cup \{\epsilon\}$,*

1. *If $u \overset{a}{\Rightarrow} u'$ then $\exists v'. v \overset{a}{\Rightarrow} v'$ s.t. $u'Rv'$; and*

2. *If $v \overset{a}{\Rightarrow} v'$ then $\exists u'. u \overset{a}{\Rightarrow} u'$ s.t. $u'Rv'$*

*Two processes $u, v$ are* weakly bisimilar

$$u \approx v$$

*if and only if they are related by a weak bisimulation*[9]. ■

and the weak bisimulation approximants $\approx_\kappa$ are defined analogously to $\sim_\kappa$ (Definition 4.3.3); namely, with $\overset{a}{\Rightarrow}$ for $\overset{a}{\rightarrow}$. The *weak bisimulation game* becomes,

**Definition 4.5.2** *$G^\tau_\kappa(u_1, u_2)$, the* weak bisimulation game up to $\kappa$, *is played between I and II, analogously to $G_\kappa$ §4.3.3.* ■

Note, in $G(u, v)$ we allow empty moves to be played, but they cannot change the state of the game, $u \overset{\epsilon}{\rightarrow} v \implies u = v$. In $G^\tau(u, v)$, *every* move may admit silent transitions. A standard theorem:

**Theorem 4.5.1 ($G^\tau$)** *$u \approx v$ iff II has a winning strategy on $G^\tau(u, v)$, and $u \approx_\kappa v$ iff II has a winning strategy on $G^\tau_\kappa(u, v)$.*

**Example 4.8 ($u \overset{\epsilon}{\Rightarrow} v \overset{\epsilon}{\Rightarrow} u$)**

1. *Trivially, $u_1 \overset{\epsilon}{\rightarrow} u_2 \implies u_1 = u_2$;*

2. *With silent actions, if $u_1 \overset{\epsilon}{\Rightarrow} u_2$ and $u_2 \overset{\epsilon}{\Rightarrow} u_1$ then $u_1 \approx u_2$, since whichever move I plays, $u_i \overset{a}{\Rightarrow} u'_i$, II can respond with $u_{3-i} \overset{\epsilon}{\Rightarrow} \overset{a}{\Rightarrow} u'_i$.*

[9]Also known as *observation equivalence*, [Mil89], and as $\tau$-bisimulation equivalence in [JJ85].

$$u \approx_{s_3} v \text{ and } v \approx_{s_3} s \text{ but } u \not\approx_{s_3} s$$

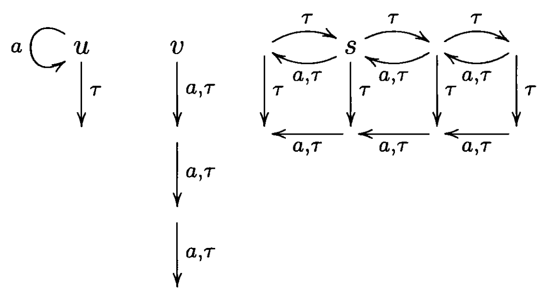


Figure 4.7: The single-step weak approximants are not transitive

**Example 4.9 (single-step weak bisimulations)** *Let a binary relation $R$ be a* single step weak bisimulation *when $uRv$ implies that, for $a \in \Sigma \cup \{\epsilon\}$,*

1. *$u \xrightarrow{a} u'$ implies $\exists v'.v \stackrel{a}{\Rightarrow} v'$ with $u'Rv'$, $u \xrightarrow{\tau} u'$ implies $\exists v'.v \stackrel{\epsilon}{\Rightarrow} v'$ with $u'Rv'$; and*

2. *$v \xrightarrow{a} v'$ implies $\exists u'.u \stackrel{a}{\Rightarrow} u'$ with $u'Rv'$, $v \xrightarrow{\tau} v'$ implies $\exists u'.u \stackrel{\epsilon}{\Rightarrow} u'$ with $u'Rv'$*

*Two processes are single-step weakly bisimilar, $u \approx_s v$ if and only if they are related by a single-step weak bisimulation.*

*That is to say, a move for player I is a strong transition, a move for Player II is a weak one – and yet, it is a standard result that $\approx\ = \ \approx_s$. A winning strategy for I on $\approx_s$ gives I a strategy for $\approx$; conversely, a winning strategy on $\approx$ can be used by I on $\approx_s$. If $u \approx^!_\kappa v$ we continue by induction on $\kappa$. A move $u \stackrel{a}{\Rightarrow} u'$ can be decomposed into $u \xrightarrow{\tau} u_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} u_l \xrightarrow{a} u_{l+1} \xrightarrow{\tau} \ldots \xrightarrow{\tau} u'$. Make each of these transitions in turn; II's responses are $v \stackrel{\epsilon}{\Rightarrow} \stackrel{\epsilon}{\Rightarrow} \ldots \stackrel{a}{\Rightarrow} \stackrel{\epsilon}{\Rightarrow} \ldots \stackrel{\epsilon}{\Rightarrow} v'$. If $u' \approx_s v'$ then (by hypothesis) in the game $u \approx v$ to $I : u \stackrel{a}{\Rightarrow} u'$ II can answer $v \stackrel{a}{\Rightarrow} v'$; hence, $u' \not\approx_s v'$.*

*It is worth noting that their respective approximants do not coincide: the $\approx_{s_\kappa}$ relations are not even equivalences (see Figure 4.7).*

Note that the correspondence between (completed) trace equivalence and bisimilarity on (normed) deterministic processes (Lemma 4.3.1) does not hold between weak trace equivalence and weak bisimilarity. Figure 4.8 shows two normed, deterministic processes which both have the weak trace language $\{a, b\}$ but are not weakly bisimilar. (The inclusions $\approx\ \subsetneq\ \equiv_{\tau LT}, \equiv_{\tau L}$ are still valid.)

The $T$ operator is a tool for showing the correspondence of strong and weak bisimilarity. A graph with $\tau$ transitions is transformed into a graph with $\stackrel{a}{\Rightarrow}$ arrows:

**Definition 4.5.3 ($T$)** *For $L = (S, \Sigma, \rightarrow)$, $T(L) = (S, \Sigma, \Rightarrow)$, where for $a \in \Sigma$, $u \stackrel{a}{\Rightarrow} v \iff u \xrightarrow{\tau^*} \xrightarrow{a} \xrightarrow{\tau^*} v$, and $u \stackrel{\epsilon}{\Rightarrow} v \iff u \xrightarrow{\tau^*} v$.* ■

$$\begin{tikzcd}
X \arrow[d, "\tau"'] \arrow[dr, "a"] & Y \arrow[d, bend right, "a"'] \arrow[d, bend left, "b"] \\
{} \arrow[r, "b"] & \epsilon
\end{tikzcd}$$

Figure 4.8: $X \equiv_{\tau L} Y$ but $X \not\approx Y$

If $u$ is a finitely-branching process, $T(u)$ need not be (Figure 4.9). We find the folklore result,

**Lemma 4.5.1 ($T$)**

1. $u \approx v \iff T(u) \sim T(v)$
2. $u \approx_\kappa v \iff T(u) \sim_\kappa T(v)$

An easy consequence of Lemma 4.5.1 is that weak bisimilarity on finite processes is decidable.

$$\begin{tikzcd}
u \arrow[r, bend left, "\tau"] & {} \arrow[l, bend left, "a"] \arrow[r, bend right, "\tau"'] & {} \arrow[l, bend right, "a"'] \arrow[r, bend left, "\tau"] & {} \arrow[l, bend left, "a"] \arrow[r, bend right, "\tau"'] & \cdots \arrow[l, bend right, "a"']
\end{tikzcd}$$

Figure 4.9: $u$ is finitely branching, $T(u)$ branches infinitely

With silent actions, any process reachable from $(L, u)$ can potentially be reached in a single transition. If the vertex-set of $L$ is $S(L)$, then

$$|S(L)| < \aleph \implies u \text{ is a sub-}\aleph\text{-branching process} \tag{4.19}$$

Since the set of reachable processes from a finitely-branching process is countable, we find that

**Lemma 4.5.2 ($T$ on finitely-branching processes)** *If $u$ is a finitely-branching process then $T(u)$ is sub-$\aleph_1$-branching.*

As a corollary, on the finitely-branching processes,

$$\approx \; = \; \approx_{\omega_1} \tag{4.20}$$

## 4.6 Other notions of weak bisimilarity

With only strong transitions, there is no concept of *livelock* – either a process can act, or it has no transitions at all: (dead)locking means termination.

$$\text{void}_\tau(u) \quad \text{:iff} \quad \forall a \in \Sigma . u \overset{a}{\not\twoheadrightarrow} \tag{4.21}$$

A process is *livelocked* when it can only act silently – if $u$ is livelocked it is void$_\tau$ but not void. Weak bisimilarity cannot distinguish between *live* and *dead*locking – between, in computing terms, a hanging system, and one which has been switched off[10]. A more subtle point levelled against weak bisimilarity is illustrated in Figure 4.10: $u$ and $v$ are weakly bisimilar, yet their branching structure differs: $v$ has the additional capability of choosing to do only a $b$ action after the first transition. *Branching bisimilarity* was developed to ensure that the *intermediate* steps of equivalent processes are matched, and thereby be more sensitive to their branching structure.

**Definition 4.6.1 (branching bisimulation)** *A symmetric relation $R$ is a* branching bisimulation *when, if $uRv$ then*

1. *If $u\xrightarrow{\tau}u'$ then $u'Rv$; and*

2. *If $u\xrightarrow{a}u'$ then $v\overset{\epsilon}{\Rightarrow}\xrightarrow{a}v'$ with $u'Rv'$*

*Processes are branching bisimilar if and only if they are related by a branching bisimulation.*[11] ■

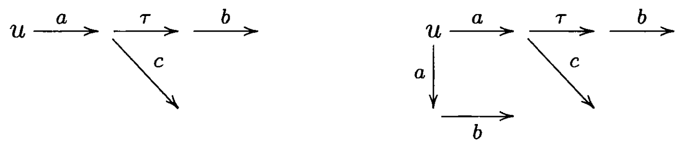


Figure 4.10: Two weakly bisimilar processes which are not branching bisimilar

Two other approaches to silence, lying in between weak and branching bisimilarity, and mutually incomparable, are the *delay bisimulations* of [Mil81b], and the $\eta$-bisimulations of [BvG87]. Each is a refinement of Condition 2; the first changes it to,

$$u\xrightarrow{a}u' \implies \exists v',v''.v\overset{\epsilon}{\Rightarrow}\xrightarrow{a}v'\overset{\epsilon}{\Rightarrow}v'' \quad u'Rv' \text{ and } u'Rv'' \tag{4.22}$$

and for $\eta$-bisimilarity,

$$u\xrightarrow{a}u' \implies \exists v',v''.v\overset{\epsilon}{\Rightarrow}v'\xrightarrow{a}\overset{\epsilon}{\Rightarrow}v'' \quad uRv' \text{ and } u'Rv'' \tag{4.23}$$

(See [vGW96] for more on this quartet.)

[10] If $u\xrightarrow{\tau}u$ and void($v$), $u \approx v$.

[11] The proof that this does in fact constitute an equivalence relation has an interesting history, [Bas96].

$$\begin{tikzcd}
 & \text{Branching bisimilarity, [GW89]} \arrow[dl] \arrow[dr] & \\
\text{Delay bisimilarity, [Mil81a]} \arrow[dr] & & \eta\text{-bisimilarity, [BvG87]} \arrow[dl] \\
 & \text{Weak bisimilarity} &
\end{tikzcd}$$

Figure 4.11: Bisimilarity-like relations incorporating silent actions, [vGW96]

Weak bisimilarity can itself be cast in these terms, see Example 4.9. It is the least observant of processes' branching, but since our criterion is the expression of *behaviour*, not fidelity to graph structure, we can return to $u$ and $v$ of Figure 4.10 and ask: is there a reason why they should be distinguished on the basis of their behaviour?

Branching-, Delay- and $\eta$-bisimilarity

$$\begin{tikzcd}
u \arrow[r, dash] \arrow[d, "\tau"] & v \arrow[dl, dash] & v \arrow[r, dash] \arrow[d, "\tau"] & u \arrow[dl, dash] \\
u' & & v' &
\end{tikzcd}$$

Branching bisimilarity

$$\begin{tikzcd}
u \arrow[r, dash] \arrow[d, "a"] & v \arrow[d, Rightarrow, "\epsilon"] & v \arrow[r, dash] \arrow[d, "a"] & u \arrow[d, Rightarrow, "\epsilon"] \\
u' \arrow[dr, dash] & {} \arrow[d, "a"] & v' \arrow[dr, dash] & {} \arrow[d, "a"] \\
& v' & & u'
\end{tikzcd}$$

Delay bisimilarity

$$\begin{tikzcd}
u \arrow[r, dash] \arrow[d, "a"] & v \arrow[d, Rightarrow, "\epsilon"] & v \arrow[r, dash] \arrow[d, "a"] & u \arrow[d, Rightarrow, "\epsilon"] \\
u' \arrow[dr, dash] \arrow[ddr, dash] & {} \arrow[d, "a"] & v' \arrow[dr, dash] \arrow[ddr, dash] & {} \arrow[d, "a"] \\
& v' \arrow[d, Rightarrow, "\epsilon"] & & u' \arrow[d, Rightarrow, "\epsilon"] \\
& v'' & & u''
\end{tikzcd}$$

$\eta$-bisimilarity

$$\begin{tikzcd}
u \arrow[r, dash] \arrow[d, "a"] \arrow[dr, dash] & v \arrow[d, Rightarrow, "\epsilon"] & v \arrow[r, dash] \arrow[d, "a"] \arrow[dr, dash] & u \arrow[d, Rightarrow, "\epsilon"] \\
u' \arrow[ddr, dash] & v' \arrow[d, "a"] & v' \arrow[ddr, dash] & u' \arrow[d, "a"] \\
& {} \arrow[d, Rightarrow, "\epsilon"] & & {} \arrow[d, Rightarrow, "\epsilon"] \\
& v'' & & u''
\end{tikzcd}$$

Weak bisimilarity (equivalent definition)

$$\begin{tikzcd}
u \arrow[r, dash] \arrow[d, "a"] & v \arrow[dd, Rightarrow, "a"] & v \arrow[r, dash] \arrow[d, "a"] & u \arrow[dd, Rightarrow, "a"] \\
u' \arrow[dr, dash] & & v' \arrow[dr, dash] & \\
& v' & & u'
\end{tikzcd}$$

Figure 4.12: Four approaches to silence (to be read left to right, top to bottom)

# Chapter 5

# Processes from context-free grammars

A *grammar* or *formal rewriting system* is a means to generate complex, potentially infinite systems from the exhaustive application of a finite set of rules. Classically these systems have been *languages*, sets of sequences over an alphabet ([Cho56, Cho57]; though they could be trees [GS97], or terms over an algebra [DJ90, Klo90], or even formalisms of biological systems, [Lin68, Pru90]). With symbols $a, b, c, X$ and the rule that, in any string of symbols, an $X$ may be rewritten either to $b$ or $aXb$, we may generate the language $\{a^n b c^n \mid n \in \mathbb{N}\}$ (Figure 5.1).

$$X \to b \quad X \to aXc$$

$$\begin{tikzcd}
X \arrow[r] \arrow[d] & aXc \arrow[r] \arrow[d] & aaXcc \arrow[r] \arrow[d] & aaaXccc \arrow[r] \arrow[d] & \cdots \\
b & abc & aabcc & aaabccc &
\end{tikzcd}$$

Figure 5.1: A context-free grammar, and its transition system rooted at $X$

This is an example of a *context-free grammar* (Type 2 in Chomsky's Hierarchy, see Figure 1.4); a distinction is made between terminal (alphabet) symbols $\Sigma$ and non-terminal or variable symbols $V$, with only the latter being rewritten. Each transition rule allows a single variable to be substituted by a sequence of variables and terminal symbols; one cannot restrict the applicability of one rule based upon the applicability of another (as the enabledness of a Petri Net transition may depend on multiple places); there is no synchronisation, no communication. The *language* of such a grammar is written on the leaves of its transition system.

**Theorem 5.0.1 (Language equivalence on context-free grammars)** *It is undecidable whether a context-free grammar over $\Sigma$ can generate $\Sigma^*$ (the* universality problem*), [Gre68]; and (as a corollary) it is undecidable whether two context-free grammars generate the same language.*

## 5.1 GNF grammars as processes

It is a fundamental result of formal language theory that any context-free grammar can be effectively transformed, preserving its language, into *Greibach Normal Form*: a grammar whose transition rules $\rightarrow$ are of the form $\rightarrow \subseteq V \times \Sigma \times V^*$ ([Gre65]). Its original application was in proving the equivalence of context-free grammars and pushdown automata; for our purposes, the interest lies in producing *labelled* transition systems (§4.1; an approach initiated by Caucal, [Cau92]). In *GNF*, each transition is of the form $X \rightarrow a\alpha$, where $X \in V$ and $\alpha \in V^*$. It is a small step to recast this as,

$$X \xrightarrow{a} \alpha \tag{5.1}$$

from, "$X$ becomes $a\alpha$" to "$X$ performs the action $a$, and becomes $\alpha$" (see [Esp01] for *grammars as processes*, and [MSS04] for a more technical view). In Figure 5.2, the language of $M$ is the set of sequences of edge labels from $M$ to $\epsilon$; it equals $\{a^n bc^c \mid n \in \mathbb{N}\}$.

$$\begin{array}{ccc} M & \xrightarrow{\text{a}} & MC \\ M & \xrightarrow{\text{b}} & \epsilon \\ C & \xrightarrow{\text{c}} & \epsilon \end{array}$$

$$\begin{tikzcd}
M \arrow[r, "a"] \arrow[d, "b"] & MC \arrow[r, "\text{a}"] \arrow[d, "\text{b}"] & MCC \arrow[r, "\text{a}"] \arrow[d, "\text{b}"] & \cdots \\
\epsilon & C \arrow[l, "\text{c}"'] & CC \arrow[l, "\text{c}"'] & \cdots \arrow[l, "\text{c}"']
\end{tikzcd}$$

Figure 5.2: A context-free grammar in GNF

### 5.1.1 Basic Process Algebra

Let $\mathcal{A} = (V, \Sigma, \rightarrow)$ be a GNF grammar. A BPA *process* is a sequence of variables, $\alpha \in V^*$; the transition relation $\rightarrow$ is extended by the rule,

$$X \xrightarrow{a} \gamma \implies X\alpha \xrightarrow{a} \gamma\alpha \tag{5.2}$$

Figure 5.2 is an example. Only the head variable of the process sequence is capable of performing an action (by which we will argue that BPA is marginally less context-free than its parallel cousin, BPP). It was introduced in [BK85] as a simple model of sequential processes. Since they generate exactly the context-free languages, trace equivalence §4.2.1 is, as a corollary of Theorem 5.0.1, undecidable (as is failures equivalence, Definition 4.2.3, and indeed every notion of equivalence of van Glabbeek's hierarchy (Figure 1.2) coarser than bisimilarity, [GH94]).

Bisimilarity (§4.3) was first proved decidable for its normed subset [Cau90, BBK93, Gro92], and then for full BPA, [CHS95] (subsequently a polynomial-time algorithm for normed BPA has been found, [HJM96a]). But whether weak bisimilarity (§4.5) is decidable remains open[1]; while it is decidable on totally normed BPA, [Hir96], even a non-trivial bound on its level of approximant collapse (§4.4.2) has yet to be found (see §5.7 and Equation 5.42).

### 5.1.2 Basic Parallel Processes

In §2.3 we went from *sequences* to *multisets* (see Definition 2.3.2, the Parikh homomorphism) – from languages to commutative languages. A BPP is a GNF grammar $\mathcal{A} = (V, \Sigma, \longrightarrow)$ in which concatenation is commutative,

$$\forall \alpha, \beta \in V^*. \alpha\beta = \beta\alpha \tag{5.3}$$

$\longrightarrow$ is extended to BPP processes by the same rule as for BPA (Equation 5.2), but with the effect that any variable of the process can act; there is no notion of leftmost derivation, or ability to arbitrarily postpone the enabledness of a variable (i.e. the BPA process $X^nY$ has to remove $n$ instances of $X$ before it can perform an action from $Y$, while as a BPP process $Y$ can act immediately; in this, BPP is more context-free than BPA: only the presence or absence of a variable determines whether its transitions can be made). An equivalent view takes BPP processes as multisets of variables, with $\longrightarrow$ extended, for $\alpha, \beta \in V^{\otimes}$, by

$$\alpha \xrightarrow{a} \beta \quad :\text{iff} \quad X \in \alpha, X \xrightarrow{a} \gamma \text{ and } (\alpha - \{X\}) \cup \gamma = \beta \tag{5.4}$$

Clearly, every BPP process over $V$ can be represented as a $|V|$-vector of natural numbers, and so are examples of $k$-tuple process ($k = |V|$, Definition 4.1.6).

**Example 5.1 (BPP)** *A two-variable BPP and the LTS generated from its process M (see Figure 5.2),*

$$\begin{aligned} M &\xrightarrow{a} MC \\ M &\xrightarrow{b} \epsilon \\ C &\xrightarrow{c} \epsilon \end{aligned}$$

$$\begin{tikzcd}
M \arrow[r, bend left, "a"] \arrow[d, "b"] & MC \arrow[l, bend left, "c"] \arrow[r, bend left, "a"] \arrow[d, "b"] & MCC \arrow[l, bend left, "c"] \arrow[r, bend left, "a"] \arrow[d, "b"] & \cdots \arrow[l, bend left, "c"] \\
\epsilon & C \arrow[l, "c"'] & CC \arrow[l, "c"'] & \cdots \arrow[l, "c"']
\end{tikzcd}$$

[1] *Branching bisimilarity* (§4.6) was proven decidable for the *totally normed BPA* in [Hüt91b], and a PSPACE algorithm for normed BPA was given in [CHT95]; there does not appear to have been much research interest in broadening this result.

BPP was introduced by Christensen in his thesis [Chr93] as a simple model of parallel processes. Where a BPA can be viewed as a single control-state Push-down automaton, BPPs are effectively communication-free Petri Nets ([Hir94b]; see §2.5.3); the applicability of much Petri Net theory has proven useful (e.g. §5.5).

Christensen shows in [Chr93] that a BPP can generate a non-context-free language. His example is

$$A\xrightarrow{a}\epsilon, B\xrightarrow{b}\epsilon, C\xrightarrow{c}\epsilon, X\xrightarrow{a}BCX, BC, X\xrightarrow{b}ACX, AC, X\xrightarrow{c}ABX, AB \tag{5.5}$$

Then, $L(X) = \{w \in \{a,b,c\} \mid w(a) = w(b) = w(c)\}$, so $L' = L(X) \cap a^*b^*c^* = \{a^nb^nc^n \mid n \geq 1\}$. But $L'$ is not context-free (using the Pumping Lemma [HU87]), yet the intersection of two context-free languages must be context-free, therefore $L(X)$ is not context-free. He goes on to provide a Pumping Lemma for BPPs,

**Lemma 5.1.1 (BPP Pumping Lemma)** *Let $\alpha$ be a BPP process, and $L = L(\alpha)$. Then there is a constant $n$ such that if $w \in L$ with $|w| > n$ then there exist $u, v, s \in \Sigma^*$ such that,*

1. *$w = us$;*
2. *$|v| \geq 1$; and*
3. *$\forall i.uv^is \in L$.*

He uses it to show that the BPP languages are disjoint from the context-free languages, using the example $\{a^nb^n \mid n \geq 1\}$. Trace (language) equivalence was shown to be undecidable by Hirshfeld, [Hir93].

**Example 5.2 (maximal finite norms)** *Let $\mathcal{A} = (V, \Sigma, \rightarrow)$ be a normed BPP whose variables are ordered according to non-decreasing norm, $V = \{X_1, \ldots, X_k\}$, $i < j \implies norm(X_i) \leq norm(X_j)$. $M(i) =_{def} \max\{norm(X_j) \mid j \leq i\}$; of course, $M(X_1) = 1$, and if $m_i = \max\{|\gamma| \mid X_i\xrightarrow{a}\gamma\}$ then $M(X_i) \leq m_i.M(X_{i-1}) + 1$. If it requires $n$ bits to define $\mathcal{A}$, then $M(X_k)$ can be written in $O(n)$ bits (and all of $M(X_1), \ldots, M(X_k)$ in $O(n^2)$).*

For bisimilarity, decidability was proven in [CHM93], and for its normed subset a polynomial algorithm was found in [HJM96b] (though without an explicit degree). PSPACE-hardness was proved in [Srb02c] (by a reduction from QSAT [Pap94, GW99]; Mayr previously found a co-NP-hardness result [May00a] using 3-SAT) and -completeness by Jančar in [Jan03]; a proof which also furnished a

$O(n^3)$-time decision procedure for normed BPP [JK04]. Mayr's co-NP-hardness result means that it is extremely unlikely that a polynomial time decision procedure exists for full BPP (as it would imply that P = NP)[2].

### 5.1.3 Bisimilarity between BPA and BPP processes

BPA and BPP generate identical classes of commutative languages, but incomparable classes of non-commutative languages; likewise, modulo bisimilarity their classes of transition system are incomparable (see [BCMS01], from which the following theorem and proof is drawn).

**Theorem 5.1.1 (BPP and BPA are incomparable)** *There exists a BPP process which is not bisimilar to any BPA process, and vice versa.*

**Proof:** Consider $M$ of Example 5.1, and imagine that $\beta$ is a BPA process with $M \sim \beta$. Let $n$ be very large, and $X\beta$ be the BPA process which corresponds to $MC^n$, with $\text{norm}(X) + \text{norm}(\beta) = n + 1$, and $\text{norm}(X) = k$. The result of performing $k$ norm-reducing transitions from $X\alpha$ must be $\alpha$, yet from $MC^n$ we can reach the two non-bisimilar states, $MC^{n-k}$ and $C^{n-(k-1)}$.

Conversely, imagine that $\alpha$ is a BPP process bisimilar to $M$ of Figure 5.2. $\text{norm}(\alpha) = 1$, hence $\alpha = X$, and $L(X) = \{a^n bc^n \mid n \in \mathbb{N}\}$. Again, let $n$ be large, then

$$X \xrightarrow{a^n} Y\gamma \xrightarrow{b} \beta\gamma \xrightarrow{c^n} \epsilon \tag{5.6}$$

where $Y \xrightarrow{b} \beta$. $\gamma \neq \epsilon$ ($\text{norm}(Y\gamma) = n + 1 > \text{norm}(Y)$), so $\gamma \xrightarrow{c^k} \epsilon$ and $\beta \xrightarrow{c^{n-k}} \epsilon$ ($k > 0$). But then, $X \xrightarrow{a^n} Y\gamma \xrightarrow{c^k} Y \xrightarrow{c} \beta \xrightarrow{c^{n-k}} \epsilon$, in which case the two processes are not even trace equivalent (see Example 4.2). □

Decidability between an arbitrary BPA and BPP is nontrivial – the union of BPA and BPP is a broader class than the finite processes (see e.g. Figure 5.4). Both Blanco and Cerna, Kretínský and Kučera found positive decidability results for normed BPA and normed BPP, [Bla95, ČKK96] (the former based on [CM90]), but it took until 2003 for decidability between full BPA and BPP to be proven (by Jančar, Kučera and Moller, [JKM03]). Their technique involved converting the BPP process (if possible) into a one-counter automata, and then using the (non-elementary, [Sén98]; EXPTIME-hard, [KM02a]) decision procedure for PDA. (A result

[2]While it can be decided in polynomial time whether a place is unbounded, Mayr shows that the problem of whether this place matters – that changing he number of tokens on the place actually changes its behaviour – is NP-hard. For example, add a variable $A \xrightarrow{a,b,c} A$ to the BPP of Figure 5.1, then if a process has $A$ enabled it does not matter, for the purposes of bisimilarity, what or how many other variables are enabled: $A \sim AM^nC^m$ (for all $n, m$). $A$ (in Mayr's terminology) *masks* the other variables.

subsequently refined by Jančar, Kot and Sawa, converting the BPP into a normal form – with a potential exponential increase in size – then directly into a BPA, [JKS05]). A key part of the proof is an innovative expression of bisimilarity on BPP processes using Jančar's *distances-to-disablings* functions, which we will cover in more depth later, §5.3.4, extend in Chapter 6, and apply in Chapter 7.

### 5.1.4 Unary languages

A language over a single-letter alphabet is termed *unary*. Observe that if $|\Sigma| = 1$, the language and commutative language (Example 2.7) of a process coincide, and moreover it makes no difference to the language whether the process is treated as a BPP or a BPA. By extending the proof of Theorem 2.4.4 (using the BPP pumping Lemma 5.1.1) we can find:

**Theorem 5.1.2 (unary context-free process languages)**

1. *All languages generated by context-free processes over a unary alphabet are regular;*
2. *Any BPP or BPA over a unary alphabet can be effectively translated into a finite process which generates the same language; and*
3. *Language equivalence on unary context-free processes is decidable.*

**Proof:** 1. This is essentially a special case of Parikh's Theorem, [Par66, Gin66]. (Esparza gives an alternative proof of which in [Esp97]). 2. It is easy to make this procedure constructive; rather than consider all strings generated by $\alpha$, explore the string-space, keeping track of which pure pumping sequences we come across and ensuring never to repeat one. 3. Language equivalence is decidable on finite processes. □

### 5.1.5 PDA control-state bisimilarity hierarchy

Modulo language equivalence, increasing the number of control-states a pushdown automaton is allowed improves succinctness of expression but does not increase the class of what can be expressed: the single-control state PDAs (i.e. BPAs) already generate all of the context-free languages ([Gre65]). In contrast, with reference to *bisimilarity*, additional control states enable the expression of strictly richer labelled transition systems.

**Lemma 5.1.2 (PDA control-state hierarchy)**
*There exists a $k$ control-state PDA with a position which is not bisimilar to any position of a PDA with fewer than $k$ control states.*

$$P = \{p_1, p_2, \ldots, p_k\} \quad \Sigma = \{n, s, a_1, \ldots, a_k\} \quad \Gamma = \{A\}$$

$$\begin{array}{rl} \forall i & (p_i, A) \xrightarrow{s} (p_i, AA) \\ \forall i & (p_i, A) \xrightarrow{a_i} (p_i, \epsilon) \\ \forall i < k & (p_i, A) \xrightarrow{n} (p_{i+1}, A) \end{array}$$

$$\begin{tikzcd}
(p_1, A) \arrow[r, shift left, "s"] \arrow[d, "n"] & (p_1, A^2) \arrow[l, shift left, "a_1"] \arrow[r, shift left, "s"] \arrow[d, "n"] & (p_1, A^3) \arrow[l, shift left, "a_1"] \arrow[r, shift left, "s"] \arrow[d, "n"] & (p_1, A^4) \arrow[l, shift left, "a_1"] \arrow[r, shift left, "s"] \arrow[d, "n"] & \cdots \arrow[l, shift left, "a_1"] \\
(p_2, A) \arrow[r, shift left, "s"] \arrow[d, "n"] & (p_2, A^2) \arrow[l, shift left, "a_2"] \arrow[r, shift left, "s"] \arrow[d, "n"] & (p_2, A^3) \arrow[l, shift left, "a_2"] \arrow[r, shift left, "s"] \arrow[d, "n"] & (p_2, A^4) \arrow[l, shift left, "a_2"] \arrow[r, shift left, "s"] \arrow[d, "n"] & \cdots \arrow[l, shift left, "a_2"] \\
\cdots \arrow[d, "n"] & \cdots \arrow[d, "n"] & \cdots \arrow[d, "n"] & \cdots \arrow[d, "n"] & \\
(p_k, A) \arrow[r, shift left, "s"] & (p_k, A^2) \arrow[l, shift left, "a_k"] \arrow[r, shift left, "s"] & (p_k, A^3) \arrow[l, shift left, "a_k"] \arrow[r, shift left, "s"] & (p_k, A^4) \arrow[l, shift left, "a_k"] \arrow[r, shift left, "s"] & \cdots \arrow[l, shift left, "a_k"]
\end{tikzcd}$$

Figure 5.3: A $k$-control state PDA

**Proof:** *Sketch*, consider the $k$-state PDA of Figure 5.3. It builds a counter of arbitrary size, and needs to keep track of whether this counter decrements on an $a_1$, $a_2$, etc.; behaviour which can be switched by a single action. That is, the stack symbols encoding the counter require $k$ different interpretations, meaning $k$ control states. □

## 5.2 Extensions of context-free processes

BPA$_\delta$ adds a special *deadlock* symbol, $\delta$, where $\forall \alpha . \delta\alpha \sim \epsilon$; [MS98] shows that this generates a strictly richer class of transition systems (but not of languages) than BPA, but $\sim$ remains decidable ([Bos96]); the latter because $\delta$ can be simulated by a fresh variable/action $D \xrightarrow{d} D$, the former because $\delta$ enables a BPA process of any norm to reach a void process in a single step.

BPP$_\tau$ allows CCS-style communication within processes. For each symbol $a$ of the alphabet there is a corresponding anti-$a$, $\overline{a}$. $\tau$ here represents the internal (and thereby *silent*) synchronisation of $a$ and anti-$a$,

$$X \xrightarrow{a} \alpha \text{ and } Y \xrightarrow{\overline{a}} \beta \implies XY\gamma \xrightarrow{\tau} \alpha\beta\gamma \tag{5.7}$$

Bisimilarity is decidable on BPP$_\tau$, but becomes undecidable if we add a unary *restriction* operator ($\alpha/L$ behaves exactly as $\alpha$, but cannot perform any action from

$L$, or $\bar{a}$, if $a \in L$), as this algebra can mimic a two-counter Minsky machine ([Min67]), and hence is Turing complete (the proof is in [Chr93], based on [Tau89]).

*Lossiness* and other forms of unreliability are an active research interest (not least because a wholly loss-free communication system is a practical impossibility), e.g. Lossy Channel Systems ([Sch01]) and Lossy Vector Addition Systems ([BM99]). Lossy BPP was introduced in [May03b], and bisimilarity proved decidable in [Srb02a] (see §5.3.3). Here, lossiness is modelled using a special action, drop $\notin \Sigma$, which allows processes to *drop* any number of variables,

$$X_1^{x_1} \dots X_k^{x_k} \xrightarrow{\text{drop}} X_1^{y_1} \dots X_k^{y_k} \qquad \forall y_1 \leq x_1, \dots, y_k \leq x_k . \exists i . y_i < x_i \tag{5.8}$$

**Example 5.3 (a lossy BPP)**
*The lossy BPP defined by $X \xrightarrow{a} XB, B \xrightarrow{b} \epsilon$ is not bisimilar to any BPP process $\alpha$. (If $\alpha$ is a BPP process, and $n$ is greater than the number of transitions offered to $\alpha$, then $\alpha \not\sim XB^n$.)*

One could alternatively define lossiness as a special usage of silent actions: for each variable $X$, add a transition $X \xrightarrow{\tau} \epsilon$ (and allow no other $\tau$ transitions); every action $\alpha \overset{a}{\Rightarrow} \alpha'$ may potentially involve the loss of variables. Call these the *silently lossy BPP*; it is easily seen that weak bisimilarity is decidable on these (unbounded, but finitely-branching) processes: $\approx$ is semidecidable by Lemma 5.5.2, and $\not\approx$ by Lemma 4.3.4.

## 5.3 Deciding bisimilarity on context-free processes

The problem of bisimilarity on context-free processes has fuelled the development of a number of powerful techniques – an arsenal which has, as yet, had little impact on the related question of weak bisimilarity. We gloss the most important of them here.

### 5.3.1 Caucal bases

A method for semideciding strong bisimilarity on classes of processes for which bisimilarity is a congruence (§2.3.1) is to show that they admit a finite *Caucal base* – a notion introduced by Caucal (and called a *self-bisimulation*) in [Cau90] to prove decidability on normed BPA (§5.1.1).

**Definition 5.3.1 (Caucal base)** *A binary relation $R$ on a class of processes is a* Caucal base *iff $uRv$ implies,*

1. *If $u \xrightarrow{a} u'$ then $\exists v' . v \xrightarrow{a} v'$ s.t. $u' \overset{R}{\equiv} v'$; and*

*2. If $v \xrightarrow{a} v'$ then $\exists u'. u \xrightarrow{a} u'$ s.t. $u' \overset{R}{\equiv} v'$*

*(See Definition 2.3.4.)* ■

**Lemma 5.3.1 (Caucal base)** *On BPP or BPA processes, $\overset{R}{\equiv} \subseteq \sim$.*

**Proof:** We wish to prove that $\overset{R}{\equiv}$ is a bisimulation. First, the *distance* or *inference depth* of $\alpha, \beta$ from $R$ is zero when $\alpha R \beta$, one when $\alpha = \alpha_1\alpha_2, \beta = \beta_1\beta_2$ and $\alpha_1 R \beta_1 \wedge \alpha_2 R \beta_2$, and so forth. Clearly, $\alpha, \beta$ are a finite distance from $R$ iff $\alpha \overset{R}{\equiv} \beta$; the proof is a simple induction on this distance. □

**Corollary 5.3.1 (Caucal base)** *$\alpha \sim \beta$ iff $\alpha R \beta$ for some Caucal base $R$.*

**Lemma 5.3.2 (finite Caucal bases)** *If $R$ is a finite relation on BPA or BPP processes, it is semidecidable whether $R$ is a Caucal base.*

**Proof:** Let $R_0$ be the pairs of processes of distance 0 from $R$ (i.e. $R_0 = R$),

$$R_{n+1} =_{\text{def}} \{(\alpha_1\alpha_2, \beta_1\beta_2) \mid \alpha_1 R_i \beta_1 \wedge \alpha_2 R_j \beta_2, i, j \leq n\} \tag{5.9}$$

If $R$ is finite, each $R_n$ can be constructed; $R$ is a bisimulation base when for each $\alpha R \beta$, any move $\alpha \xrightarrow{a} \alpha'$ has a counterpart $\beta \xrightarrow{a} \beta'$ with $\exists n. \alpha' R \beta'$ (and the dual, $\beta \xrightarrow{a} \beta'$). If they exist, we can find them. □

The existence of a finite Caucal base for normed BPA was shown in [Cau90] (and though decidability was already known, his proof is significantly easier to understand than that of [BBK87, BBK93]); a finite base for full BPA was found by Christensen, Hüttle and Stirling in [CHS92, CHS95]. Hirshfeld proved in [Hir94a] that a finite base exists for BPP (a result related to Theorem 2.3.1). See [BCMS01] for details.

### 5.3.2 Hirshfeld trees

Otherwise known as *expansion trees*, and first proposed in [Hir94c]; see e.g. [JM99].

**Definition 5.3.2 (expansion)** *A finite binary relation $A$ on BPP processes is* expanded *by $A'$ when,*

1. *$\alpha A \beta$ and $\alpha \xrightarrow{a} \alpha'$ implies that $\beta \xrightarrow{a} \beta'$ with $\alpha' A' \beta'$;*
2. *$\alpha A \beta$ and $\beta \xrightarrow{a} \beta'$ implies that $\alpha \xrightarrow{a} \alpha'$ with $\alpha' A' \beta'$; and*
3. *Minimality: no proper subset of $A'$ satisfies conditions 1 and 2*

■

**Definition 5.3.3 (Hirshfeld tree)** *An* expansion tree *for $(\alpha, \beta)$ is constructed:*

1. *Its root is labelled $\{(\alpha, \beta)\}$;*
2. *The children of a node $A$ are all expansions $A'$ of $A$, with the provisos that:*
   (a) *Every pair $(\alpha, \alpha)$ is omitted; and*
   (b) *Every pair $(\alpha, \beta)$ which has already occurred in $A$ or its ancestors is omitted.*

*The $\emptyset$-labelled leaves are termed* successful*; a non-empty leaf (i.e. one with no expansions) is* unsuccessful*. A branch is successful if and only if it does not terminate with an unsuccessful node.* ■

**Lemma 5.3.3 (expansion trees)** *$\alpha \sim \beta$ iff the expansion tree grown from $(\alpha, \beta)$ has a successful branch.*

Using a notion – domination, §3.1 – which we will employ in §5.7.1, one can eliminate the possibility of infinite branches. A finite Hirshfeld tree may be constructed, and a successful branch sought.

### 5.3.3 Tableau proofs

Tableau proofs are prominent in modal logic and proof theory ([Fit96, DGHP99]); when written they resemble *natural deductions* ([BE99, Gir89]), finite tree-like structures which decompose a statement to-be-proved until its truth or falsehood becomes trivial. A *tableau* is a finite collection of rules describing how a sentence (in our case, an expression of a process) may be broken down into atomic propositions. It is *sound* when it cannot prove anything that is false, and *complete* when everything true has a successful tableau.

$$\frac{\text{Goal}}{\text{Subgoal}_1, \ldots, \text{Subgoal}_n} \text{ side conditions} \tag{5.10}$$

Tableau techniques were first applied to infinite labelled transition systems in [BS90] (extending the work of [CES86]) in order to verify temporal properties of processes (*safety*: nothing bad will happen; *liveness*: something good will eventually happen, etc), and have since furnished a number of decidability proofs for equivalence relations on context-free processes (e.g. strong bisimilarity on BPP [R9̈5], BPA [Hüt91a], normed pushdown processes [Sti98a]; branching bisimilarity on BPA [Hüt91b]). In [Srb02a], Srba presents a class of transition systems called the Effective Commutative Transition Systems together with a sound and complete tableau, enabling him to prove at a stroke the decidability of bisimulation equivalence on BPP, lossy BPP, BPP with interrupt and timed-arc BPP (§5.2).

**Definition 5.3.4 (very simple BPP)** *A BPP algebra is* very simple *iff for all* $t, s \in \rightarrow$, $l(t) = l(s) \implies t = s$. ■

As a toy example, we can demonstrate the decidability of bisimulation equivalence on the *very simple BPP* (in which each transition is uniquely identified by its action name) with a sound and complete two rule tableau[3].

First, a variable $X$ is *non-removable*, $\textsc{nr}(X)$, when

$$\textsc{nr}(X) \quad \text{:iff} \quad X \xrightarrow{a} \alpha \implies \alpha(X) > 0 \tag{5.11}$$

our two rules are,

$$\frac{X\alpha = X\beta}{\alpha = \beta}\neg\textsc{nr}(X) \qquad \frac{X^{n+1}\alpha = X^{m+1}\beta}{\alpha = \beta}\textsc{nr}(X), \alpha(X) = \beta(X) = 0 \tag{5.12}$$

For any pair of $\alpha, \beta$ of very simple BPP, the exhaustive application of the above two variable-cancelling rules will create a necessarily finite tree, whose leaves are of the form $\gamma = \delta$ (where neither rule applies). It is not difficult to see that $\alpha \sim \beta$ if and only if the tree (or tableau) rooted at $\alpha = \beta$ has leaves labelled by $\epsilon = \epsilon$.

### 5.3.4 Jančar's distances-to-disablings

In [Jan03], Jančar presents a novel method for capturing bisimilarity on BPP processes (further developed in [JKM03]). We will return in depth to the general technique in Chapter 7, using rather different notation and a very different approach. For now, with distance defined as per Equation 6.1 (page 78),

$$\mathrm{dist}(u, v) =_{\mathrm{def}} \min\{|w| \mid u \xrightarrow{w} v\} \tag{5.13}$$

the distances-to-disablings functions are constructed: for any action $a \in \Sigma$,

$$dd_a(\alpha) =_{\mathrm{def}} \min\{\mathrm{dist}(\alpha, \beta) \mid \beta \overset{a}{\not\rightarrow}\} \tag{5.14}$$

is a disabling function; and, if $\mathcal{F} = (d_1, \ldots, d_l)$ is a sequence of already constructed distances-to-disablings functions ($d_i : V^* \rightarrow \mathbb{N} \cup \{-1, \omega\}$), and $\delta = (\delta_1, \ldots, \delta_l) \in$

[3]The *very simple grammars* originated with [But72]; see also [KH66].

$(\mathbb{N} \cup \{-1, \omega\})^l$ then

$$dd_{(a,\mathcal{F},\delta)}(\alpha) \quad =_{\text{def}} \quad \min\{\text{dist}(\alpha,\beta) \,|\, \forall i.d_i(\beta) < \omega \text{ and} \tag{5.15}$$

$$\forall \beta'.\beta \xrightarrow{a} \beta', \mathcal{F}(\beta') - \mathcal{F}(\beta) \neq \delta\} \tag{5.16}$$

where,

$$(d_1, \ldots, d_l)(\alpha) = (d_1(\alpha), \ldots, d_l(\alpha)) \tag{5.17}$$

and the subtraction in Line 5.16 is pointwise. Either $\beta$ cannot make an $a$ move, or if it can there is at least one $d_i$ function whose *change*, upon making the move, is *not* $\delta_i$. (By convention, $\min \emptyset = \omega$.) We will reintroduce (generalisations of) these functions in Chapter 6 (with what I hope is more intuitive notation). As noted in [JKM03], on finitely branching processes, $u, v$,

$$u \sim v \quad \text{iff} \quad d(u) = d(v) \text{ for all distances-to-disablings functions } d \tag{5.18}$$

(see Lemma 6.2.1 for a generalisation). Our immediate interest is the tight and surprising correspondence between distances-to-disablings and the $\text{NORM}(Q)$ functions. If $Q \subseteq V$ is a set of variables, the *norm relative to* $Q$ of a process is the minimal number of transitions require to remove every instance of a variable from $Q$:

$$\text{NORM}(Q) =_{\text{def}} \lambda\alpha.\min\{|w| \,|\, \alpha \xrightarrow{w} \alpha' \wedge \forall X \in Q.\alpha'(X) = 0\} \tag{5.19}$$

In particular, $\text{NORM}(\emptyset)(\alpha) = 0$ for all $\alpha$, and $\text{NORM}(V) = \text{norm}$. (Recalling NR of Equation 5.11, page 68, $\text{NR}(X)$ iff $\text{NORM}(\{X\})(X) = \omega$.) Two processes $\alpha, \beta$ are NORM-equivalence when $\forall Q.\text{NORM}(Q)(\alpha) = \text{NORM}(Q)(\beta)$; in fact, NORM-equivalence coincides with distances-to-disablings equivalence – i.e., with bisimilarity. In [Jan03], Jančar goes further, providing a PSPACE decision procedure:

**Lemma 5.3.4 (NORM)** *For any BPP system $\mathcal{A} = (V, \Sigma, \rightarrow)$, we can effectively find, in space that is polynomial in the size of $\mathcal{A}$, a finite number of sets of places, $Q_1, \ldots, Q_m \subseteq V$ such that,*

$$\forall \alpha, \beta \in V^*, \quad \alpha \sim \beta \quad \textit{iff} \quad \forall 0 < i \leq m.\text{NORM}(Q_i)(\alpha) = \text{NORM}(Q_i)(\beta) \tag{5.20}$$

In the four years since, no (published) progress has been made in applying the theory to weak bisimilarity. This will become clearer in Chapter 7.

## 5.4 Weak bisimilarity

Weak bisimilarity is EXPTIME-hard on BPA [May02][4], and is at least PSPACE-hard on BPP. Context-free processes permitted a silent action may branch infinitely, Figure 5.4; the straightforward exploration means of semideciding inequivalence of §4.3.4 no longer applies, and creates problems that have remained open, even for normed processes.

$$\begin{tikzcd}
 & P \arrow[r, "\tau"] \arrow[d, "\tau"] & A \arrow[loop right, "a"] & & \\
\epsilon & Q \arrow[l, "a"] \arrow[r, shift left, "\tau"] & QQ \arrow[l, shift left, "a"] \arrow[r, shift left, "\tau"] & QQQ \arrow[l, shift left, "a"] \arrow[r, shift left, "\tau"] & \cdots \arrow[l, shift left, "a"]
\end{tikzcd}$$

$P\xrightarrow{\tau}A, P\xrightarrow{\tau}Q$
$Q\xrightarrow{\tau}QQ, Q\xrightarrow{a}\epsilon$
$A\xrightarrow{a}A$

Figure 5.4: Context-free processes $P, Q$ where $P \approx^!_\omega Q$

Mayr shows in [May02] that weak bisimilarity is undecidable for BPA with a control unit of size 2. In [KRS06], Křetínský, Řehák and Strejček broaden this result to BPP: they define a special (monotonic) case of finite control unit extended BPP and BPA, *fc*BPA and *fc*BPP, and prove that weak bisimilarity is undecidable even for normed *fc*BPA and normed *fc*BPP.

Stříbrná and Cerná have attempted without much success to apply weak variants of Hirshfeld trees (§5.3.2) and Caucal bases (§5.3.1) to weak bisimilarity on BPA, [SČ02].

The first tractable algorithms involving weak bisimilarity on infinite state systems have been found by Kučera and Mayr, who show that weak bisimilarity between BPA and finite processes, and between normed BPP and finite processes, is decidable in polynomial time, [KM02b].

**Lemma 5.4.1 (congruence)** *Weak bisimilarity $\approx$ and the weak bisimulation approximants $\approx_\kappa$ are congruences on BPP processes (with reference to process concatenation).*

**Proof:** $R = \{(\alpha\delta, \beta\delta) \mid \alpha \approx \beta\}$ is a weak bisimulation. Let $\alpha\gamma R \beta\gamma$, and wlog I: $\alpha\gamma \stackrel{a}{\Rightarrow} \alpha'\gamma'$. Either $\alpha \stackrel{a}{\Rightarrow} \alpha'$ and $\gamma \stackrel{\epsilon}{\Rightarrow} \gamma'$ or $\alpha \stackrel{\epsilon}{\Rightarrow} \alpha'$ and $\gamma \stackrel{a}{\Rightarrow} \gamma'$; in the former case, $\exists \beta . \beta \stackrel{a}{\Rightarrow} \beta'$ with $\alpha' \approx \beta'$ by assumption, but then $\alpha'\gamma' R \beta'\gamma'$ (the other cases are similar). For the weak approximants, $\alpha \approx_\kappa \beta \implies \forall\gamma . \alpha\gamma \approx_\kappa \beta\gamma$, by induction on $\kappa$. Let $\alpha \approx_\kappa \beta$

[4]At the end of [Hir96], Hirshfeld writes,

We [...] conjecture that eventually it will be shown that weak (and branching) bisimulation is decidable. The work that was needed to prove the result for the very restricted case of totally normed BPA [...] is discouraging.

and $I : \alpha\gamma \overset{a}{\Rightarrow} \alpha'\gamma'$, where wlog $\alpha \overset{a}{\Rightarrow} \alpha'$ and $\gamma \overset{\epsilon}{\Rightarrow} \gamma$. By assumption $\forall\mu < \kappa\ \exists\beta'.\beta \overset{a}{\Rightarrow} \beta'$ with $\alpha' \approx_\mu \beta'$, and by hypothesis $\alpha'\gamma' \approx_\mu \beta'\gamma'$. □

### 5.4.1 Totally normed BPP

The first non-trivial subclass of BPP for which $\approx$ was proven decidable is the *totally normed BPP* [Hir96][5], which strengthens the normed condition to, for every $X \in V$,

$$0 < \mathrm{norm}_\tau(X) < \infty \tag{5.21}$$

While these processes may branch infinitely, they remain *finitely approximable*, in that the approximant hierarchy collapses at $\omega$:

$$\alpha \not\approx \beta \implies \exists n.\alpha \not\approx_n \beta \tag{5.22}$$

(Let $\alpha \not\approx \beta$, and $\alpha \overset{a}{\Rightarrow} \alpha'$ be $I$'s move. $II$ can have only finitely many responses $\beta \overset{a}{\Rightarrow} \beta'$ with $\mathrm{norm}_\tau(\alpha') = \mathrm{norm}_\tau(\beta')$.) See Theorem 5.6.1.

### 5.4.2 Purely-generated BPP

The first non-finitely approximable subset of BPP for which we have a decidability proof is the *normed, purely generated BPP*, as given by Stirling in [Sti01c].

A variable $X \in V$ is a *generator* when it can be used to build, in a single weak transition, an arbitrary number of variables. Define,

$$G(X) =_{\mathrm{def}} \{A \mid X \overset{\epsilon}{\Rightarrow} XA \wedge \mathrm{norm}_\tau(A) = 0\} \tag{5.23}$$

It is a *pure* generator when,

$$X \overset{\epsilon}{\Rightarrow} \alpha \wedge \mathrm{norm}_\tau(X) = \mathrm{norm}_\tau(\alpha) \implies \alpha = X\alpha' \tag{5.24}$$

(I.e. every $\mathrm{norm}_\tau$-neutral silent transition is either *generating* or leaves the process unchanged). The tableau proof is difficult, and while the paper states that this work could be extended to the normed BPP, "the combinatorics become awesome."

## 5.5 Weak bisimilarity is semidecidable on BPP

Since $\approx \subseteq (\mathbb{N}^{|V|})^2$ is a congruence on BPP (Lemma 5.4.1), and $\mathbb{N}^{|V|}$ is finitely generated (Example 2.9), Theorem 2.4.1 applies, and we find that $\approx$ is itself a semilinear relation (Corollary 2.4.1). There are countably many such relations, so given an

[5]The definition of *totally normed* owes to Hüttel, [Hüt91b].

effective means to test whether a candidate relation is, in fact, a weak bisimulation, would give us its semidecidability. For, if $\alpha \approx \beta$, then there exists a weak bisimulation $R \subseteq \approx$ with $\alpha R \beta$. Semilinear set membership is decidable (Lemma 2.4.1); it remains to show that we can test whether a relation is a weak bisimulation. For this, we define a semilinear encoding of the $\overset{a}{\Rightarrow}$ relation (from [Esp97], which is in turn based on [Jan95b]):

Recall the relation Reach of §2.5.3. Projecting away the $\tau$ transitions gives the effective semilinear relation,

$$\text{Reach}_\tau =_{\text{def}} \{(\vec{\alpha}, \vec{\sigma}, \vec{\beta}) \mid \alpha \overset{\sigma}{\Rightarrow} \beta\} \tag{5.25}$$

and, applying Theorem 2.4.3,

**Lemma 5.5.1** ($\phi_a$) *We can, for any BPP, effectively construct a formula of Presburger Arithmetic (§2.4.1) $\phi_a$ such that, for all of its processes $\alpha, \beta$,*

$$\alpha \overset{a}{\Rightarrow} \beta \quad \iff \quad \phi_a[\vec{\alpha}, \vec{\beta}] \tag{5.26}$$

**Proof:** Let $\psi[\vec{\alpha}, \vec{\sigma}, \vec{\beta}]$ iff $\text{Reach}_\tau(\vec{\alpha}, \vec{\sigma}, \vec{\beta})$, then

$$\phi_a(\vec{\alpha}, \vec{\beta}) \equiv \exists \vec{\sigma}.\psi(\vec{\alpha}, \vec{\sigma}, \vec{\beta}) \text{ and } \Sigma_i \vec{\sigma}(i) = 1 \land \vec{\sigma}(i) = 1 \implies X_i \overset{a}{\Rightarrow}$$

(where of course whether an atom can perform a $\overset{a}{\Rightarrow}$ transition can be Presburger encoded). □

Finally, if $\rho$ is the Presburger equivalent of a candidate weak bisimulation, define the Presburger formula,

$$\begin{aligned}
\chi_\rho \quad =_{\text{def}} \quad & \forall \vec{\alpha}, \vec{\beta}.\rho(\vec{\alpha}, \vec{\beta}) \implies & (5.27)\\
& \forall a, \vec{\alpha}'.\psi_a(\vec{\alpha}, \vec{\alpha}') \implies \exists \vec{\beta}'.\psi_a(\vec{\beta}, \vec{\beta}').\rho(\vec{\alpha}', \vec{\beta}') \land & (5.28)\\
& \forall a, \vec{\beta}'.\psi_a(\vec{\beta}, \vec{\beta}') \implies \exists \vec{\alpha}'.\psi_a(\vec{\alpha}, \vec{\alpha}').\rho(\vec{\alpha}', \vec{\beta}') & (5.29)
\end{aligned}$$

and find:

**Lemma 5.5.2** ($\approx$ **on BPP**) $\approx$ *is semidecidable on BPP.*

**Proof:** If $\alpha \approx \beta$, enumerate all Presburger equivalents $\rho$ of semilinear relations on $\mathbb{N}^{2.|V|}$, testing for each:

1. $\rho[\vec{\alpha}, \vec{\beta}]$; and

2. $\chi_\rho$

eventually we must find a relation which satisfies both, and may conclude that, indeed, $\alpha \approx \beta$. □

## 5.6 The finite weak approximants

Stříbrná gives in [Stř99] a neat encoding of the finite weak approximants into formulas of Presburger Arithmetic,

$$\alpha \approx_n \beta \quad \Longleftrightarrow \quad \phi_n[\vec{\alpha}, \vec{\beta}] \tag{5.30}$$

where,

$$\begin{aligned}
\phi_0(\vec{\alpha}, \vec{\beta}) &\equiv \mathit{True} && (5.31)\\
\phi_{n+1}(\vec{\alpha}, \vec{\beta}) &\equiv \forall \vec{\alpha}', a.\psi_a(\vec{\alpha}, \vec{\alpha}') \implies && (5.32)\\
&\qquad \exists \vec{\beta}'.\psi_a(\vec{\beta}, \vec{\beta}') \wedge \phi_n(\vec{\alpha}', \vec{\beta}') && (5.33)\\
&\qquad\quad \text{and} && (5.34)\\
&\quad \forall \vec{\beta}', a.\psi_a(\vec{\beta}, \vec{\beta}') \implies && (5.35)\\
&\qquad \exists \vec{\alpha}'.\psi_a(\vec{\alpha}, \vec{\alpha}') \wedge \phi_n(\vec{\alpha}', \vec{\beta}') && (5.36)
\end{aligned}$$

which gives us their decidability (Theorem 2.4.2), and moreover the decidability of $\approx$ on the finitely-approximable subclass of BPP (the totally normed BPP §5.4.1, for example). However, no complexity bounds are given, and it is difficult to see how this method could be extended to $\approx_\omega$.

**Theorem 5.6.1 (the finitely approximable BPP)** *On any finitely-approximable class of BPP, $\approx$ is decidable.*

**Proof:** Semidecidability is given by Lemma 5.5.2; to semidecide non-equivalence, decide each finite approximant in turn. □

Lemma 4.3.5 puts a bound on the number of equivalence classes there are modulo $\approx_n$: the players of $G_n^\tau(\alpha, \beta)$ have infinitely many choices, but each comes from finitely many equivalence classes. On BPP we find a stronger result, and a second way to decide the finite weak approximants.

Define the *capping* of (natural numbers) $x$ by $n$ to be,

$$\frac{n}{x} =_{\text{def}} \begin{cases} x & \text{if } x \leq n \\ n & \text{otherwise} \end{cases} \tag{5.37}$$

and extend this to monomials by, $\frac{n}{X_1^{x_1} \ldots X_k^{x_k}} =_{\text{def}} X_1^{\frac{n}{x_1}} \ldots X_k^{\frac{n}{x_k}}$. It's easy to see that,

$$\alpha \approx_1 \beta \text{ iff } \frac{1}{\alpha} \approx_1 \frac{1}{\beta} \tag{5.38}$$

i.e. to decide $\alpha \approx_1 \beta$ on processes of any size, it suffices to test the question on a pair of processes composed of at most $k$ variables a piece.

**Lemma 5.6.1 (BPP finite approximant branching)** *For BPP processes over* $V = \{X_1, \ldots, X_k\}$,

$$\alpha \approx_n \beta \textit{ iff } \frac{k^n}{\alpha} \approx_n \frac{k^n}{\beta} \tag{5.39}$$

**Proof:** *Sketch,* we wish to show that,

$$\alpha \approx_n \frac{k^n}{\alpha} \tag{5.40}$$

An induction on $n$; let $\beta = \frac{k^{n+1}}{\alpha}$, and consider the game $G^{\tau}_{n+1}(\alpha, \beta)$. If $I : \alpha \overset{\epsilon}{\Rightarrow} \alpha'$, where $\alpha = \underbrace{X_1 \ldots X_1}_{x_1} \underbrace{X_2 \ldots X_2}_{x_2} \ldots \underbrace{X_k \ldots X_k}_{x_k}$, first decompose its transition sequence:

$$\begin{tikzcd}[row sep=large]
\alpha = & X_1 \arrow[d, Rightarrow, "\epsilon"] & X_1 \arrow[d, Rightarrow, "\epsilon"] & \cdots & X_1 \arrow[d, Rightarrow, "\epsilon"] & X_2 \arrow[d, Rightarrow, "\epsilon"] & \cdots\cdots & X_k \arrow[d, Rightarrow, "\epsilon"] \\
\alpha' = & \gamma_{1_1} & \gamma_{1_2} & \cdots & \gamma_{1_{x_1}} & \gamma_{2_1} & \cdots\cdots & \gamma_{k_{x_k}}
\end{tikzcd}$$

next, consider each $X_i$ in turn. There are $x_i$ instances of $X_i$ in $\alpha$; in $\alpha'$ they become $\gamma_{i_1}, \gamma_{i_2}, \ldots, \gamma_{i_{x_i}}$ (where of course it could be that $\gamma_{i_j} = X_i$). If $x_i \leq k^{n+1}$, or

$$|\{\gamma_{i_j} \mid \gamma_{i_j} \neq \epsilon, 1 \leq j \leq x_i\}| \leq k^{n+1} \tag{5.41}$$

there is nothing to be done, as the transition sequence can be replicated by $\beta$. Otherwise, we have at least $k^{n+1}$ components, each of which involves at least one variable; by hypothesis it makes no difference to $\approx_n$ whether $\alpha'$ has $k^n$ instances of a variable or more than $k^n$ instances: subsequent $X \overset{\epsilon}{\Rightarrow} \gamma_{i_j}$ transitions are redundant. The pigeon-hole principle implies that this transition sequence, minus the redundant parts, can also be met by $\alpha'$. (The cases $\alpha \overset{a}{\Rightarrow} \alpha'$ and $\beta \overset{a}{\Rightarrow} \beta'$ are similar.) □

(Note, this seems a very conservative bound – finding an example $\alpha$ for which $\alpha \not\approx_2 \frac{2}{\alpha}$, say, is difficult.) Given any $\alpha, \beta$, we can use Lemma 5.6.1 to effectively construct two finite processes $u, v$ with the property that, $\alpha \approx_n \beta$ iff $u \sim v$.

**Corollary 5.6.1 ($\approx_n$ as $\sim$ between finite processes)** *On BPP processes, $\approx_n$ can be effectively reduced to $\sim$ on finite processes.*

## 5.7 Approximant collapse

Given two parallel processes $P, Q$ with $P \approx^!_\omega Q$ (Figure 5.4), we can for any $n \in \mathbb{N}$ find processes $X, Y$ where $X_n \approx^!_{\omega+n} Y_n$, by adding $2n$ fresh variables,

$$\begin{array}{ll} X_0 = P & Y_0 = Q \\ X_{i+1} \xrightarrow{a} X_i & Y_{i+1} \xrightarrow{a} Y_i \end{array}$$

Can one go further, do there exist BPP processes whose inequivalence cannot be seen by $\approx_{\omega\times 2}$? In the *sequential* case (as first noted by Stříbrná in [Stř99]) the level of approximant collapse must be at least $\omega^\omega$. For any $\kappa < \omega^\omega$ we wish to construct a pair of BPA processes $\alpha, \beta$ with $\alpha \not\approx \beta$ but $\alpha \approx_\kappa \beta$. Let $V = \{X_0, \ldots, X_{n-1}\}$, $\Sigma = \{a, \tau\}$, and

$$X_0 \xrightarrow{a} \epsilon \qquad X_i \xrightarrow{\tau} \epsilon \qquad X_{i+1} \xrightarrow{\tau} X_{i+1} X_i \tag{5.42}$$

For each $\kappa < \omega^n$, with Cantor Normal Form (Theorem 2.1.1)

$$\kappa = \omega^{n-1} a_{n-1} + \ldots + \omega a_1 + a_0 \tag{5.43}$$

let $\alpha_\kappa = X_0^{a_0} \ldots X_{n-1}^{a_{n-1}}$. It can be proven that $\kappa < \mu < \omega^n$ implies $\alpha_\kappa \approx^!_\kappa \alpha_\mu$ (the following comes from [HMS06]). In the following, $\alpha, \beta, \gamma$ are interpreted sequentially as BPA processes.

- If $\alpha \approx \beta$ then

  1. $\gamma\alpha \approx \gamma\beta$; and
  2. $\alpha\gamma \approx \beta\gamma$

  (of course, for BPA processes $\alpha\gamma$ does not in general equal $\gamma\alpha$), with the caveat that for 2 to hold, $\alpha \approx \epsilon \implies \alpha \overset{\epsilon}{\Rightarrow} \epsilon$.

- If $i > j$ then $X_i X_j \approx X_i$.
- As a consequence, for every $\beta \in V^*$ there exists an $\alpha_\kappa$ such that $\alpha_\kappa \approx \beta$.
- Next, for every $\kappa \leq \omega^k$, $X_k \overset{\epsilon}{\Rightarrow} \alpha_\kappa$.
- For every $\mu \leq \kappa$, $\alpha_\kappa \overset{\epsilon}{\Rightarrow} \alpha_\mu$.
- If $\alpha_\kappa \overset{\epsilon}{\Rightarrow} \beta$ then $\exists \mu \leq \kappa . \alpha_\mu \approx \beta$, and
- If $\alpha_\kappa \overset{a}{\Rightarrow} \beta$ then $\exists \mu < \kappa . \alpha_\mu \approx \beta$.

Then, for any ordinal $\omega^k$ we can find a pair of processes from a $k+1$-variable BPA which are $\approx_{\omega^k}$ equivalent, but are not weakly bisimilar. Namely $\alpha_{\omega^k}$ and $\alpha_{\omega^{k+1}}$. It is a simple transfinite induction to verify that the game $G^{\tau}_{\omega^k}(\alpha_{\omega^k}, \alpha_{\omega^{k+1}})$ is won by $II$; a winning strategy for $I$ on $G^{\tau}_{\omega^k+1}(\alpha_{\omega^k}, \alpha_{\omega^{k+1}})$ begins with $\alpha_{\omega^{k+1}} \overset{a}{\Rightarrow} \alpha_{\omega^k}$. For sequential context-free processes, the conjecture is:

**Conjecture 5.7.1 (Stříbrná)** *On BPA processes,* $\approx = \approx_{\omega^\omega}$.

Returning to *parallel* context-free processes, a long-standing conjecture holds otherwise:

**Conjecture 5.7.2 (Hirshfeld, Jančar)** *On BPP processes,* $\approx = \approx_{\omega\times 2}$.

BPP processes branch finitely, hence Lemma 4.4.4 gives us,

$$\approx = \approx_{\omega_1} \tag{5.44}$$

We can do a little better: in [Stř99] there is an argument owing to Julian Bradfield to show that the BPP approximant hierarchy collapses by level ${\omega_1}^{CK}$, the first non-recursive ordinal. Yet this is number of a completely different order to $\epsilon_0$, let alone to $\omega \cdot 2$; we will progress to a more modest bound, one which relies on the analysis of Dickson's Lemma given in Chapter 3.

### 5.7.1 An $\omega^\omega$ bound for BPP

(Presented at CSL 2006, and published in LICS 2006, [HMS06].) First, a result based on a lemma of Hirshfeld's [Hir96],

**Lemma 5.7.1 (domination)** *If* $\alpha \approx^!_\kappa \beta$ *and* $\alpha\gamma \approx^!_\mu \beta\delta$ *for some* $\mu < \kappa$, *then*

$$\alpha\gamma \approx^!_\mu \alpha\delta \quad \text{and} \quad \beta\gamma \approx^!_\mu \beta\delta$$

*Furthermore,* $(\alpha\gamma, \alpha\delta) <_{lex} (\beta\gamma, \beta\delta)$ *or* $(\beta\gamma, \beta\delta) <_{lex} (\alpha\gamma, \alpha\delta)$, *where* $<_{lex}$ *is the usual lexicographic ordering.*

**Proof:** The weak approximants $\approx_\kappa$ on BPP are congruences (Lemma 5.4.1): $\alpha\delta \approx_\kappa \beta\delta \approx_\mu \alpha\gamma$; however, if $\alpha\gamma \approx_{\mu+1} \alpha\delta$ then $\alpha\gamma \approx_{\mu+1} \alpha\delta \approx_\kappa \beta\delta$ (the other case is similar). Note that $\alpha \neq \beta$, so either $\alpha <_{lex} \beta$ or $\beta <_{lex} \alpha$. □

**Lemma 5.7.2** *For BPP processes* $\alpha, \beta$, $h(omt(\alpha, \beta)) = \kappa$ *implies the existence of a* $\mathbb{N}^{|V|\times 2}$*-labelled non-dominating tree of height* $\kappa$.

**Proof:** The *level* of a node is its distance from the root. Apply the following substitution method to each successive level $i$ of $t = omt(\alpha, \beta)$: for all level $i$ nodes $u$, if $u$ dominates an ancestor $v$,

$$l(v) = (\phi, \psi) \qquad l(u) = (\phi\gamma, \psi\delta) \tag{5.45}$$

*replace* $u$ in $t$ by $u' = omt(\phi\gamma, \phi\delta)$ if $(\phi\gamma, \phi\delta) <_{\text{lex}} (\psi\gamma, \psi\delta)$ and by $u' = omt(\psi\gamma, \psi\delta)$ if $(\psi\gamma, \psi\delta) <_{\text{lex}} (\phi\gamma, \phi\delta)$. Lemma 5.7.1 means this is a height-preserving operation. If $u'$ also dominates an ancestor, repeat the process. That $<_{\text{lex}}$ is well-founded guarantees that it can be repeated at most a finite number of times (for each branch); and that $t$ is well-founded means that there are a finite number of levels to cover. □

**Theorem 5.7.1 (BPP approximant collapse)**
*On BPP processes,* $\approx = \approx_{\omega^\omega}$

**Proof:** Assume $\alpha \not\approx \beta$ but $\alpha \approx_{\omega^\omega} \beta$. By Lemma 5.7.2 there exists a non-dominating vector-labelled tree of height $\kappa > \omega^\omega$, but Theorem 3.3.1 implies $\kappa < \omega^\omega$. □

# Chapter 6

# Distances-to-disablings

The distances-to-disablings functions were developed in [Jan03] to address the problem of strong bisimilarity on Basic Parallel Processes (for which there exists a pleasing correspondence with their relativised norms, §5.3.4). At the end of his paper Jančar conjectures his method to be a promising approach to the (still open) problem of weak bisimilarity, but work on this appears to have stalled, with no papers published on the subject in the years since. We begin by defining a generalised version of distances-to-disablings (with simplified notation) applicable to all general processes, with details as to the connection with approximant collapse and the finite approximants. Finding that their natural weak analogue immediately fails, we develop a form which is applicable to weak bisimilarity; and that, with promising if partial results, is the subject of Chapter 7.

## 6.1 Strong distances, and norm revisited

The *strong distance* from $u$ to $v$ is defined to be the minimal number of transitions required to reach $v$ from $u$:[1],

$$\mathrm{dist}(u,v) =_{\mathrm{def}} \min\{|w| \mid u \xrightarrow{w} v\} \tag{6.1}$$

For a set $V$ of processes, we write $\mathrm{dist}(u,V) =_{\mathrm{def}} \min\{\mathrm{dist}(u,v) \mid v \in V\}$; for $P$ a predicate on processes,

$$\mathrm{dist}(u,P) =_{\mathrm{def}} \mathrm{dist}(u,\{v \mid P(v)\}) \tag{6.2}$$

[1]This distance is not intended to be a *metric* – though one of the first approaches to process equivalence was metric-space oriented, [dBZ82a, dBZ82b]. We might (in passing) look at the class of processes on which dist is reflexive, transitive and obeys the triangle inequality. A simple condition is both necessary and sufficient: for all processes $u$, $u \xrightarrow{a} v \implies \exists b. v \xrightarrow{b} u$.

(And, $\mathrm{dist}(P) =_{\mathrm{def}} \lambda u.\mathrm{dist}(u, P)$.) Recalling norm of §4.1.1, we find

$$\mathrm{norm}(u) = \mathrm{dist}(u, \mathrm{void}) = \mathrm{dist}(u, \{v \mid \forall a.v\overset{a}{\not\rightarrow}\}) \tag{6.3}$$

Rather than ask the distance until all actions are disabled, we might ask it of a particular action.

$$d_a(u) =_{\mathrm{def}} \mathrm{dist}(u, \overset{a}{\not\rightarrow}) \tag{6.4}$$

A useful notation, trailed at the end of §4.1, is to define the *change* (modulo a given function) produced by a transition (see the $\delta$s of §5.3.4). For $f$ a function on a set of processes $P$, $f : P \longrightarrow \mathbb{Z}_\infty$, $\delta \in \mathbb{Z}_\infty$, and $u, v \in P$:

$$u\xrightarrow{a}_{f,\delta}v \quad \text{:iff} \quad u\xrightarrow{a}v \tag{6.5}$$

$$\text{and } f(u) < \infty \tag{6.6}$$

$$\text{and } f(u) + \delta = f(v) \tag{6.7}$$

Where by convention, $\infty + \delta = \delta + \infty = \infty$ for any $\delta$[2]. Again, $u\xrightarrow{a}_{f,\delta}$ abbreviates $\exists u'.u\xrightarrow{a}_{f,\delta}u'$.

**Example 6.1**

1. *In Figure 4.1 (page 41), u has two transitions, $u\xrightarrow{a}u$ and $u\xrightarrow{a}u'$ (with void($u'$)). $d_a(u') = 0$, so $d_a(u) = 1$ and $u\xrightarrow{a}_{d_a,0}u$, $u\xrightarrow{a}_{d_a,-1}u'$.*

2. *In Figure 4.4 (page 44), s and t only have $d_a$-*reducing *actions: $s\xrightarrow{a}_{d_a,\delta} \Longrightarrow \delta = -1$.*

3. *In Figure 4.6 (page 47), $u\xrightarrow{a}_{d_a,n}$ and $v\xrightarrow{a}_{d_a,n}$ for every $n \geq -1$, while*

$$u\xrightarrow{a}_{d_a,\infty} \quad v\overset{a}{\not\rightarrow}_{d_a,\infty} \tag{6.8}$$

*and $s\overset{a}{\not\rightarrow}_{d_a,\delta}$ for every $\delta$ (since $d_a(s) = \infty$).*

(Note, this can be viewed as a generalisation of the norm-stratifying notation of [Sti01c], i.e. $\alpha\overset{a}{\Rightarrow}_n\beta$ iff $\alpha\overset{a}{\Rightarrow}_{\mathrm{norm},n}\beta$.) To conclude, if $F$ is a set of pairs $(f, \delta)$,

$$u\xrightarrow{a}_F v \quad \text{:iff} \quad \forall(f, \delta) \in F.u\xrightarrow{a}_{f,\delta}v \tag{6.9}$$

We are now able to elegantly ask convoluted questions like: "What is the minimum distance until we can, with an $a$-labelled transition, go from a situation in

[2]To avoid confusion, $\infty$ is used where $\omega$ might appear (in for example [Jan03]).

which the $a$ action can (eventually) be disabled to one in which it cannot."

$$\lambda u.\mathrm{dist}(u, \overset{a}{\not\rightarrow}_{d_a,\infty}) \tag{6.10}$$

The use of such questions is the subject of this Chapter.

## 6.2 DD$^{\mathcal{C}}$ functions

**Definition 6.2.1 (DD$^{\mathcal{C}}$)** *The* distances to disablings *functions are constructed: if $a \in \Sigma$, and $D$ is a (potentially empty) set of pairs $(d, \delta)$, where $d$ is an already constructed* DD$^{\mathcal{C}}$ *function and $\delta \in \mathbb{N}_{-1,\infty}$, then*

$$dd(a, D) = \lambda u. \min\{dist(u, v) \mid v \overset{a}{\not\rightarrow}_D\}$$

*is a* DD$^{\mathcal{C}}$ *function.*[3] *No restriction is put on the size of the $D$ sets.* ■

Where for convenience, $\mathbb{N}_{-1,\infty}$ abbreviates $\mathbb{N} \cup \{-1, \infty\}$. Specifically, $d_a = dd(a, \emptyset)$, and $\lambda u.\mathrm{dist}(\overset{a}{\not\rightarrow}_{d_a,\infty}) = dd(a, \{(dd(a, \emptyset), \infty)\})$ (Equation 6.10). When $D$ is a singleton set, $\{d', \delta\}$, we abbreviate $dd(a, \{d', \delta\})$ to $dd(a, d', \delta)$. The *height* of a DD$^{\mathcal{C}}$ function is defined to be the *depth* of nesting in its construction:

$$h(dd(a, D)) \quad = \quad \sup\{h(d) + 1 \mid (d, \delta) \in D\} \tag{6.11}$$

So, $h(d_a) = h(dd(a, \emptyset)) = 1$, and by construction, $h(d) \in \mathcal{O}$. In the manner of Definition 4.1.5 we restrict the *branching* of these DD trees: $d$ is a DD$^{\aleph}$ function when the cardinalities of the $D$ sets used in its construction are bounded strictly above by $\aleph$. We call DD$^{\aleph_0}$ the *finite* and DD$^{\aleph_1}$ the *countable* DD$^{\mathcal{C}}$ functions, and abbreviate the former to DD.

$$u \equiv_{\mathrm{DD}^{\mathcal{C}}} v \quad \text{:iff} \quad \forall d \in \mathrm{DD}^{\mathcal{C}}.d(u) = d(v) \tag{6.12}$$

**Lemma 6.2.1 (DD$^{\mathcal{C}}$)**

1. *For any $\aleph$ there exist processes $u, v$ with $u \equiv_{\mathrm{DD}^{\aleph}} v$ but $u \not\sim v$;*
2. *On general processes, $\sim\, = \,\equiv_{\mathrm{DD}^{\mathcal{C}}}$; and*
3. *On the sub-$\aleph$-branching processes, $\sim\, = \,\equiv_{\mathrm{DD}^{\aleph}}$.*

[3] Recall that $\mathcal{C}$ represents the class of cardinal numbers, §2.2 (page 22).

$$\begin{tikzcd}
u \arrow[d, "n"] & \begin{array}{l} d(u) \neq d(v) \\ n = \min\{d(u), d(v)\} \end{array} & v \arrow[d, "n"] \\
u' \arrow[d, "a"] & d(u') = 0 \iff d(v') \neq 0 & v' \arrow[d, "a"] \\
u'' \arrow[d, "n'"] & \begin{array}{l} d' \in D(d) \\ d'(u'') \neq d'(v'') \\ n' = \min\{d'(u''), d'(v'')\} \end{array} & v'' \arrow[d, "n'"] \\
\vdots & & \vdots \\
u''' \arrow[d, "a'"] & \begin{array}{l} d''' = dd(a', \emptyset) \\ d'''(u''') > d'''(v''') = 0 \end{array} & v''' \arrow[d, "\not{a'}"] \\
{} & & {}
\end{tikzcd}$$

Figure 6.1: An illustration of $u \not\equiv_{\mathrm{DD}^{\aleph}} v \implies u \not\sim v$

**Proof:** 1. Let $\aleph'$ be a regular cardinal with $\aleph' \geq \aleph$ (Lemma 2.2.1). We will construct a pair of processes $u_\kappa, v_\kappa$, with $u_\kappa \sim^{!}_{\kappa} v_\kappa$ but $u_\kappa \equiv_{\mathrm{DD}^{\aleph'}} v_\kappa$ (implying $u_\kappa \equiv_{\mathrm{DD}^{\aleph}} v_\kappa$). Specifically, define $u_0$ to be a void process,

$$\mathrm{void}(u_0) \tag{6.13}$$

and define, for each $\kappa \in \mathcal{O}$, $u_\kappa$ to be the process with the transitions,

$$\forall \mu < \kappa . u_\kappa \xrightarrow{a} u_\mu \tag{6.14}$$

and finally, let $v_\kappa$ be the process with transitions:

$$\forall \mu < \kappa . v_\kappa \xrightarrow{a} u_\mu \quad \text{and} \quad v_\kappa \xrightarrow{a} v_\kappa \tag{6.15}$$

It is easy to see that, for all $\kappa$, $u_\kappa \sim^!_\kappa v_\kappa$. To begin with, we will prove that for any $d \in \mathrm{DD}^{\aleph'}$,

$$\kappa' > \kappa > h(d) \implies d(u_\kappa) = d(u'_\kappa) \tag{6.16}$$

by an induction on $h(d)$. For $\kappa > \kappa' > 0$, $d_a(u_\kappa) = d_a(u_{\kappa'}) = 1$. For the inductive step, let $d = dd(a, D)$. We may assume by the induction hypothesis that,

$$\kappa' > \kappa \geq h(d) \implies D(u_\kappa) = D(u_{\kappa'}) \tag{6.17}$$

Since $d(u_0) = 0$ for all $d \in \mathrm{DD}^{\mathcal{C}}$, we find that $d(u_\kappa), d(u_{\kappa'}) \in \{0, 1\}$.

1. If $d(u_\kappa) = 1$ then there exists a $\mu < \kappa$ such that $u_\kappa \xrightarrow{a}_D u_\mu$, but then $u_{\kappa'} \xrightarrow{a}_D u_\mu$, so $d(u'_\kappa) > 0$;

2. If $d(u_{\kappa'}) = 1$ and $d(u_\kappa) = 0$ then there exists a $\kappa' > \mu \geq \kappa$ with $u_{\kappa'} \xrightarrow{a}_D u_\mu$. Since (by induction hypothesis)

$$D(u_{\kappa'}) = D(u_\mu) = D(u_\kappa) = D(u_{h(d)}) \tag{6.18}$$

it must be that $\pi_2(D) = \{0\}$, so we have $u_\kappa \xrightarrow{a}_D u_{h(d)}$, contradicting $d(u_\kappa) = 0$.

It can be shown by induction on $h(d)$ that,

$$\kappa > h(d) \implies d(u_\kappa) = d(v_\kappa) \tag{6.19}$$

This is easily shown by cases, as above. Observe that the sole transition from $v_\kappa$ which cannot be exactly replicated from $u_\kappa$ is $v_\kappa \xrightarrow{a} v_\kappa$: the only subtle case $d = dd(a, D)$ is when,

$$\pi_2(D) = \{0\} \quad \text{and} \quad d(u_\kappa) = 0 < d(v_\kappa) \tag{6.20}$$

(i.e. $v_\kappa \xrightarrow{a}_D v_\kappa$). However, we know that $D(u_\kappa) = D(u_{h(d)})$, and $u_\kappa \xrightarrow{a} u_{h(d)}$, which implies $d(u_\kappa) > 0$. To finish the proof, recall that $\aleph'$ is a regular cardinal: Theorem 2.2.1 implies that for all $d \in \mathrm{DD}^{\aleph'}$, $h(d) < \aleph'$, i.e.

$$u_{\aleph'} \not\sim v_{\aleph'} \quad \text{and} \quad u_{\aleph'} \equiv_{\mathrm{DD}^{\aleph}} v_{\aleph'} \tag{6.21}$$

2. Let $d(u) \neq d(v)$, we wish to prove that $u \not\sim v$, and will do so by transfinite induction on $h(d)$. First, if $d = d_a = dd(a, \emptyset)$ (for some $a \in \Sigma$), and wlog $c =$

$d_a(u) < d_a(v)$, then there exists a $w \in \Sigma^c$ s.t. $u \xrightarrow{w} u' \overset{a}{\not\rightarrow}$, but if $v \xrightarrow{w} v'$ then $v' \xrightarrow{a}$; hence, $u \not\sim_c v$. Next, let $d = dd(a, D)$ $(D \neq \emptyset)$ and wlog $c = d(u) < d(v)$. If $u \not\sim v$ we are already done, so assume that $u \sim v$ (this assumption will lead us to a contradiction). Again, either $u \not\sim_\omega v$, or we can force play to $u', v'$ s.t. $u' \overset{a}{\not\rightarrow}_D$ and $v' \xrightarrow{a}_D$ (and by assumption, $u' \sim v'$). If $\pi_1(D)(u') \neq \pi_1(D)(v')$ then by induction hypothesis we are done; otherwise, move $v' \xrightarrow{a}_D v''$. For any response $u' \xrightarrow{a} u''$ there must be a $(d', \delta) \in D$ s.t. $d'(v'') = d'(v') + \delta \neq d'(u') + \delta = d'(u'')$ (see Figure 6.1). Again, since play has been forced by $I$ to this situation, and $h(d') < h(d)$, we can invoke the induction hypothesis to conclude that $u \not\sim v$.

Conversely, if $u \not\sim_\kappa v$ we wish to construct a DD$^C$ function $d$ to distinguish $u$ from $v$. This proceeds by transfinite induction on $\kappa$. If $u \not\sim_1 v$ then $\exists a. u \xrightarrow{a} \iff v \overset{a}{\not\rightarrow}$, i.e. $d_a(u) = 0 \iff d_a(v) \neq 0$, so simply $d = d_a$. Otherwise, if $u \not\sim_\kappa v$, and wlog $I$'s optimal move is $u \xrightarrow{a} u'$, for each $v \xrightarrow{a} v'$ we have, by induction hypothesis, a function $d_{v'}$ which distinguishes $u'$ from $v'$. Create a set of pairs $D$ such that $\pi_1(D) = \{d_{v'} \mid v \xrightarrow{a} v'\}$ (i.e. the first components of our set of pairs $D$ are the $d_{v'}$ functions given to us by the induction hypothesis), and $\pi_2(D)$ be such that $u \xrightarrow{a}_D u'$ (i.e. the second components – the $\delta$ values – are such that $u \xrightarrow{a}_D u'$; it is worth noting that there will always exist numbers drawn from $\mathbb{N}_{-1,\infty}$ to satisfy this). Then by construction, $v \overset{a}{\not\rightarrow}_D v'$, and if $d = dd(a, D)$, $d(v) = 0 < d(u)$.

3. Observe that in the construction above, if $u$ and $v$ are sub-$\aleph$-branching, $d \in \mathrm{DD}^{\aleph}$. □

On the finitely-branching processes, $\sim\; = \;\equiv_{\mathrm{DD}}$. Immediately we find that $\equiv_{\mathrm{DD}}$ is undecidable on the Petri Nets, but decidable on Pushdown Automata and the Basic Parallel Processes. By Lemma 4.5.1, and the semidecidability of $\approx$ on BPP (Lemma 5.5.2), we find too that $\equiv_{\mathrm{DD}^C}$ is semidecidable on BPP with silent moves.

### 6.2.1 Approximant collapse

The connection with *approximant collapse* (§4.4) is a simple extension of the proof of Lemma 6.2.1.2. If $d \in \mathrm{DD}$ (a finite DD function), and $d(u) \neq d(v)$ Lemma 6.2.1 gives us $u \not\sim v$, but we can say something stronger: $u \not\sim_{\omega^2} v$. Wlog let $d = dd(a, D)$, and $c_1 = d(u) < d(v)$. Player $I$ can force play in $c_1$ steps to a pair of states $u', v'$ for which $\exists(d', \delta) \in D$ with $d(u') \neq d(v')$ and $c_2 = \min\{d(u'), d(v')\}$. This trick can be repeated at most $h(d) - 1$ many times before we arrive at $d' = d_a$. Of course, $d_a(u') \neq d_a(v') \implies u' \not\sim_m v'$, where $m = \min\{d_a(u'), d_a(v')\} + 1$.

The reason we cannot conclude $u \not\sim_{c_1+c_2+\ldots+c_n} v$ (for $h(d) = n$; i.e. $u \not\sim_\omega v$) is that the size of each $c_i$ can (potentially) be chosen by Player $II$, and each decision is (again, potentially) deferred until $c_1 + 1 + c_2 + 1 + \ldots + c_{i-1} + 1$ steps through the game. That is, $II$ has *at most* $h(d)-1$ opportunities to arbitrarily postpone the loss of $G(u,v)$: this equates to $u \not\sim_{\omega \times h(d)} v$. This result extends to the regular cardinals:

**Lemma 6.2.2 (DD$^{\aleph}$ and approximant collapse)** *If* $\sim \; = \; \equiv_{\mathrm{DD}^{\aleph}}$ *on a class of processes, for a regular cardinal* $\aleph$*, then* $\sim \; = \; \sim_{\aleph \times \omega}$ *on that class.*

**Proof:** If two processes $u, v$ are not bisimilar, and bisimilarity is captured by $\equiv_{\mathrm{DD}^{\aleph}}$, then there exists a $d \in \mathrm{DD}^{\aleph}$ where $d(u) \neq d(v)$. If $\aleph$ is regular, then $h(d) = \kappa < \aleph$, and from this we wish to conclude that $u \not\sim_{\aleph \times \omega} v$. Either $d = dd(a, \emptyset)$ $(u \not\sim_{\omega} v)$, or we can find a $d' \in D(d)$ where $h(d') < \kappa$ such that $I$ is able to force play to $u', v'$ and $d'(u') \neq d'(v')$. We find then a (not necessarily countable) sequence of DD$^{\aleph}$ functions $d, d', d'', \ldots$ with $\kappa = h(d) > h(d') > h(d'') > \ldots$ At each step in the sequence $II$ has an opportunity to arbitrarily postpone loss of the game; its length is bounded strictly above by $\aleph$, and so $u \not\sim_{\aleph \times \omega} v$. □

### 6.2.2 The finite approximants

On general processes with finite alphabets the full power of DD$^C$ is required to capture bisimilarity. We will find here that the distinguishing strength of *trace* equivalence is met already by the finite DD functions, and do so through a more general result, a sequel to §4.3.5.

**Lemma 6.2.3 (DD and $\sim_n$)** *For any processes $u, v$ over a finite alphabet and any $n \in \mathbb{N}$ we can find a finite sequence of* DD *functions $D_n$, each of height bounded by $n$, such that,*

$$D_n(u) = D_n(v) \implies u \sim_n v \tag{6.22}$$

**Proof:** We will build two finite trees, $U, V$, essentially the unfolding of $u$ and $v$ modulo $\sim_n$. The trees are rooted at $u$ and $v$ respectively; call this level $n$. At level 1, we finish. For a node $u'$ at level $m+1$, partition the transitions of $u'$ into equivalence classes modulo $\sim_{m+1}$. By Lemma 4.3.5 there are finitely many classes; choose one representative $u \xrightarrow{a} u''$ of each to be the children of $u'$.

$D_1 = (d_a)_{a \in \Sigma}$. For $D_{m+1}$, we turn to the $m$th levels of our two trees. For each $u' \xrightarrow{a}_D u''$, where $\pi_1(D) = D_m$, add $dd(a, D)$ to $D_m$. That is,

$$\begin{aligned} D_{m+1} &= D_m \cup && (6.23)\\ &\quad (dd(a,D) \mid \pi_1(D) = D_m, && (6.24)\\ &\qquad\qquad u \text{ an } m+1\text{th level node of } U \text{ or } V && (6.25)\\ &\qquad\qquad u' \text{ an } m\text{th level node and } u \xrightarrow{a}_D u') && (6.26) \end{aligned}$$

Finally, let $D_{n+1}(u) = D_{n+1}(v)$, and wlog $I : u \xrightarrow{a} u'$. There is by its construction a move $u \xrightarrow{a} u''$ recorded in $U$ with $u' \sim_n u''$; let $D$ be such that $u \xrightarrow{a}_D u''$ and $\pi_1(D) = D_n$. Since $dd(a, D)(v) = dd(a, D)(u) > 0$ (by assumption), $\exists v'. v \xrightarrow{a}_D v'$; $D_n(u) = D_n(v)$ so $D_n(u'') = D_n(v')$ – i.e. $u'' \sim_n v'$ □

**Corollary 6.2.1 (DD and $\sim_\omega$)** *For processes $u, v$ over a finite alphabet,*

$$u \equiv_{\text{DD}} v \implies u \sim_\omega v \tag{6.27}$$

To return to the opening of this section, recall that $\sim_\omega \subsetneq \equiv_L$ (Example 4.4). We find, $\equiv_L \supsetneq \sim_\omega \supsetneq \equiv_{\text{DD}}$.

We can express $\approx$ with the $DD^{\mathcal{C}}$ functions by using the $T$ operator (Definition 4.5.3). The difficulty with this approach is that while $u, v$ might branch finitely, and be expressed by the finite DD functions, $T(u), T(v)$ could branch infinitely, and require the infinite $\text{DD}^{\aleph_1}$ functions to capture bisimilarity on. The problem of capturing bisimilarity on infinitely branching systems using *finitary* disablings functions is central to this thesis.

## 6.3 Weak distances

The weak analogues of bisimilarity and bisimulation approximants (§4.5) are straightforward: swap strong arrows $\rightarrow$ for weak, $\Rightarrow$. This natural approach to *distance*[4] yields,

$$\text{dist}_{nw}(u, v) =_{\text{def}} \min\{|w| \mid u \overset{w}{\Rightarrow} v\}$$

The *natural weak distance to disabling* functions, $\text{DD}^{\mathcal{C}}_{wn}$, become, in turn,

$$dd_{nw}(a, D) =_{\text{def}} \lambda u. \min\{\text{dist}_{nw}(u, u') \mid u' \overset{a}{\not\Rightarrow}_D\} \tag{6.28}$$

for $a \in \Sigma \cup \{\epsilon\}$, and $D$ a (possibly empty) set of pairs $(d, \delta)$, for $d$ an already constructed natural weak distance to disabling function, and $\delta \in \mathbb{N}_{-1,\infty}$.

**Lemma 6.3.1** *For $d \in \text{DD}^{\mathcal{C}}_{wn}$, $u \overset{\epsilon}{\Rightarrow} v \implies d(v) \geq d(u)$.*

In the absence of $\tau$ actions these equal the $\text{DD}^{\mathcal{C}}$ functions; with them, we meet an insuperable problem: even the trivial processes $u, v$ of Figure 6.2 are beyond its power to differentiate.

(An induction on $h(d)$. If $d = dd_{nw}(b, \emptyset)$, $d(u) = d(v) = 0$. For $d = dd_{nw}(b, D)$ and $b \neq \epsilon$, $d(v) = 0 = d(u)$ (Lemma 6.3.1). Finally, let $d = dd_{nw}(\epsilon, D)$ and $d(v) = \infty$.

[4]First put into print by Hüttel in [Hüt91b].

$$\begin{tikzcd} u \arrow[r, bend left, "a"] \arrow[r, bend right, "\tau"'] & v \end{tikzcd}$$

Figure 6.2: $u \equiv_{\mathrm{DD}^{\mathcal{C}}_{nw}} v$

$\pi_1(D)(u) = \pi_1(D)(v)$ by hypothesis, and $\pi_2(D)(v) = \{0\}$ (since $v\overset{\epsilon}{\Rightarrow}_D v$). But then, $u\overset{\epsilon}{\Rightarrow}_D v$, hence $d(u) = \infty$.)

### 6.3.1 Refining weak distance

No distinction is made by $\mathrm{dist}_{nw}$ between $(u, u)$ and $(u, v)$ when $u\overset{\epsilon}{\Rightarrow}v$, which suggests we refine our notion of weak distance. A simple if ugly solution follows.

**Definition 6.3.1 ($\mathbf{dist}_\tau$)**

$$dist_\tau(u, v) = \begin{cases} 0 & \leftarrow u = v \\ dist_\tau(u, v) + 1 & \leftarrow \textit{otherwise} \end{cases}$$

■

**Definition 6.3.2 ($\mathbf{DD}^{\mathcal{C}}_\tau$)** *The* weak distances to disablings *functions are constructed: if $a \in \Sigma \cup \{\epsilon\}$, and $D$ is a (potentially empty) set of pairs $(d, \delta)$, where $d$ is an already constructed* $\mathrm{DD}^{\mathcal{C}}_\tau$ *function and $\delta \in \mathbb{N}_{-2,-1,\infty}$, then*

$$dd_\tau(a, D) = \lambda u. \min\{dist_\tau(u, v) \mid v\overset{a}{\not\Rightarrow}_D\}$$

*is a* $\mathrm{DD}^{\mathcal{C}}_\tau$ *function.* ■

Again, $d_{a_\tau} =_{\mathrm{def}} dd_\tau(a, \emptyset)$, and $dd_\tau(a, d, \delta)$ abbreviates $dd_\tau(a, \{(d, \delta)\})$.

Considering Figure 6.2 (page 86), $d_{a_\tau}(u) = 1 > d_{a_\tau}(v)$. However, transitions of the form $u\overset{a}{\Rightarrow}_{d,-2}v$ are now possible (for example, if the $\tau$ arrow is removed from the above, $u\overset{a}{\Rightarrow}_{d_{a_\tau},-2}v$). A small lemma helps to deal with this:

**Lemma 6.3.2 (ugly lemma)** *If $0 < dist_\tau(u, u'') < dist_\tau(v, v'')$ then there exists an $a \in \Sigma \cup \{\epsilon\}$ s.t. $u\overset{a}{\Rightarrow}u'$ and for all $v\overset{a}{\Rightarrow}v'$, $0 \le dist_\tau(u', u'') < dist_\tau(v', v'')$.*

**Proof:** Say $1 = \mathrm{dist}_\tau(u, u'') < \mathrm{dist}_\tau(v, v'') = 2$: $u\overset{\epsilon}{\rightarrow}u''$, but $v\overset{a}{\rightarrow}v'' \implies a \neq \epsilon$. □

The results for $\mathrm{DD}^{\mathcal{C}}$, Lemma 6.2.1, follow through without too much complication to $\mathrm{DD}^{\mathcal{C}}_\tau$:

**Lemma 6.3.3 ($\mathbf{DD}^{\mathcal{C}}_\tau$)** *On processes with a silent $\tau$ action,*

1. *For any $\aleph$ there exist processes $u, v$ with $u \equiv_{\mathrm{DD}_\tau^\aleph} v$ but $u \not\approx v$;*

2. *On general processes, $\approx = \equiv_{\mathrm{DD}^\mathcal{C}}$; and*

3. *On the sub-$\aleph_0$-branching processes, $\approx = \equiv_{\mathrm{DD}_\tau^{\aleph_1}}$. On the sub-$\aleph$-branching processes, for $\aleph$ a regular cardinal greater than $\aleph_0$, $\approx = \equiv_{\mathrm{DD}_\tau^\aleph}$.*

**Proof:**

1. The processes $u_\kappa, v_\kappa$ constructed in the proof of 6.2.1.1 serves here too.

2. Let $d(u) \neq d(v)$, with $d \in \mathrm{DD}_\tau^\mathcal{C}$. Prove by induction on $h(d)$ that $u \not\approx v$. For $d = dd(a, \emptyset)$, $d(u) \neq d(v)$ gives $I$ a winning strategy on $G_\omega^\tau(u, v)$ (Definition 4.5.2). Wlog let $c = d(u) < d(v)$; by Lemma 6.3.2 $I$ can force play (in $c - 1$ steps if $c > 1$, $c$ steps otherwise) to processes $u', v'$ with $0 = d(u') < d(v')$; then, $v' \overset{a}{\Rightarrow}$ but $u \overset{a}{\not\Rightarrow}$, hence $u \not\approx_{c+1} v$.

   Let $d = dd(a, D)$, and assume that $\exists d' \in \pi_1(D).d'(u) \neq d'(v) \implies u \not\approx v$. A winning strategy for $I$: wlog $\pi_1(D)(u) = \pi_1(D)(v)$, and $c = d(u) < d(v)$. Apply Lemma 6.3.2 to force play to $u', v'$ with $0 = d(u') < d(v')$. If $\pi_1(D)(u') \neq \pi_1(D)(v')$, we are done. Otherwise, make $v' \overset{a}{\Rightarrow}_D v''$, and by definition for any response $u' \overset{a}{\Rightarrow} u''$ there exists a $(d', \delta) \in D$ with $d'(u') + \delta \neq d'(u'')$, i.e. $d'(v'') \neq d''(u'')$, and by hypothesis $u'' \not\approx v''$.

   Conversely, if $u \not\approx_\kappa v$ we wish to construct a $\mathrm{DD}_\tau^\mathcal{C}$ function which distinguishes $u$ from $v$. This proceeds in an identical manner to the proof of Lemma 6.2.1.2.

3. The construction above requires a $(d, \delta)$ pair for each process reachable in a single $\overset{a}{\Rightarrow}$. For finitely branching processes, this reachability set is countable. If the processes $u, v$ are sub-$\aleph$-branching for a regular cardinal $\aleph > \aleph_0$, then: from $u$ (respectively, $v$) we can reach $\kappa_1 < \aleph$ processes in a single $\tau$ step, $\kappa_1 \times \kappa_2 < \aleph$ in the next step, and so on; the regularity of $\aleph$ ensures that the resultant set cannot have $\aleph$ or more elements, hence in the construction above the $\mathrm{DD}_\tau^\aleph$ functions suffice.

□

We abbreviate $\mathrm{DD}_\tau^{\aleph_0}$ to $\mathrm{DD}_\tau$.

**Example 6.2 ($\mathrm{DD}_\tau$ and processes with finite alphabets)** *It should be clear that Lemma 4.3.5 of §4.3.5 applies equally to the finite weak approximants:*

$$| \approx_n | = 2^{\left.|\Sigma| 2^{|\Sigma| 2^{|\Sigma| \cdots 2^{|\Sigma|}}}\right\}n} \tag{6.29}$$

*and moreover that the construction of §6.2.2 (suitably modified) enables us to conclude that on general processes with finite alphabets,*

$$u \equiv_{\mathrm{DD}_\tau} v \implies u \approx_\omega v \tag{6.30}$$

## 6.4 Extending $\mathrm{DD}_\tau$

$$\begin{tikzcd}
u \arrow[loop above, "a"] \arrow[d, "\tau"'] \arrow[dr, "\tau"] \arrow[drr, "\tau"] \arrow[drrr, bend left, "\tau"] & & & \\
s_0 & s_1 \arrow[l, "a{,}\tau"] & s_2 \arrow[l, "a{,}\tau"] & \cdots \arrow[l, "a{,}\tau"] \\
v \arrow[u, "\tau"] \arrow[ur, "\tau"] \arrow[urr, "\tau"] \arrow[urrr, bend right, "\tau"'] & & &
\end{tikzcd}$$

Figure 6.3: $u \approx^!_\omega v$, but $u \equiv_{\mathrm{DD}_\tau} v$

Processes $u, v$ of Figure 6.3 are *strong finitely branching* but *weak infinitely branching*, and while not weakly bisimilar, they are $\mathrm{DD}_\tau$ equivalent. We will go through the proof of this with some care (in particular point 2(c)ii), since it will reappear, with complications, for Figure 7.3.

1. $u \approx^!_\omega v$: *II* has a winning strategy for $G^\tau_n(u,v)$, for any $n$; *I* has a winning strategy on $G^\tau_\omega(u,v)$ (repeatedly move $u \overset{a}{\Rightarrow} u$).

2. $\forall d \in \mathrm{DD}_\tau$, $d(u) = d(v)$, by induction on $h(d)$. Clearly, $dd(a,\emptyset)(u) = dd(a,\emptyset)(v)$; if $d = dd(a, D)$, with $\pi_1(D)(u) = \pi_1(D)(v)$, then:

   (a) $d(u), d(v) \in \{0, 1, \infty\}$;

   (b) if $d = dd(\epsilon, \emptyset)$, and

      i. $d(u) = 0$, so $u \overset{\epsilon}{\not\Rightarrow}_D$, implying that $\pi_2(D) \neq \{0\}$ and so $v \overset{\epsilon}{\not\Rightarrow}_D v'$ for $v' \in \{v, s_0, s_1, \ldots\}$, which is to say: $d(v) = 0$.

      ii. $d(u) = 1$, so $u \overset{\epsilon}{\Rightarrow} s_j \overset{\epsilon}{\not\Rightarrow}_D$. Either $u \overset{\epsilon}{\Rightarrow}_D u$ (so $\pi_2(D) = \{0\}$ and $v \overset{\epsilon}{\Rightarrow}_D v$), or $u \overset{\epsilon}{\Rightarrow}_D s_j$, but then $v \overset{\epsilon}{\Rightarrow}_D s_j$, and $v \overset{\epsilon}{\Rightarrow} s_i \overset{\epsilon}{\not\Rightarrow}_D$: $d(v) = 1$.

      iii. $d(u) = \infty$, but then $d(s_0) = \infty$, so $s_0 \overset{\epsilon}{\Rightarrow}_D s_0$, meaning $\pi_2(D) = \{0\}$ and $v \overset{\epsilon}{\Rightarrow}_D v$: $d(v) = \infty$.

   (c) if $d = dd(a, \emptyset)$, and

      i. $d(u) = 0$, so $\forall i. u \overset{a}{\not\Rightarrow}_D s_i$, in which case $\forall i. v \overset{a}{\not\Rightarrow}_D s_i$ and $d(v) = 0$.

ii. $d(u) = 1$, so $d(u)\overset{a}{\Rightarrow}_D$. Since $s_0\overset{a}{\not\Rightarrow}$, $d(v) \leq 0$. If $d(u)\overset{a}{\Rightarrow}_D s_j$, then $v\overset{a}{\Rightarrow}_D s_j$ and we are done. Otherwise, $u\overset{a}{\Rightarrow}_D u$, so $\pi_2(D) = \{0\}$. We wish to now show that, for all $d' \in \mathrm{DD}_\tau$ we can find an $n$ such that,

$$\text{for every } i \geq n \quad d'(v) = d'(s_i) \tag{6.31}$$

in which case, there would exist an $n$ s.t. $v\overset{a}{\Rightarrow}_D s_n$, and we would have $d(v) = 1$. A second induction on $\mathrm{DD}_\tau$ height. If $d' = dd(a, D')$, let $n(d') = \max\{n(d'') \mid d'' \in D'\} + 1$; it is straightforward (in light of what has gone) to verify that $\forall i \geq n(d').d'(v) = d'(s_i)$.

iii. if $d(u) = \infty$ then $s_0\overset{a}{\Rightarrow}_D$, but $s_0\overset{a}{\not\Rightarrow}$.

Lemma 6.3.3.3 implies the existence of a countable disabling function $d \in \mathrm{DD}_\tau^{\aleph_1}$ to distinguish $u$ from $v$. We would rather keep things finite – must, if computability is to be maintained – and find that a small extension to the $\mathrm{DD}_\tau$ functions suffices:

$$d^\omega_{a_\tau}(u) = \min\{\mathrm{dist}_\tau(u, v) \mid v\overset{a^\omega}{\not\Rightarrow}\} \tag{6.32}$$

where, for ordinals $\kappa \in \mathcal{O}$,

$$u\overset{a^\kappa}{\Rightarrow}_F \quad \text{:iff} \quad \begin{cases} u\overset{a}{\Rightarrow}_F u'\overset{a^\mu}{\Rightarrow}_F & \leftarrow \kappa = \mu + 1 \\ \forall \mu < \kappa . u\overset{a^\mu}{\Rightarrow}_F & \leftarrow \text{otherwise} \end{cases} \tag{6.33}$$

I.e. $u\overset{a^\omega}{\Rightarrow}$ iff $\forall n . u\overset{a^n}{\Rightarrow}$ (moreover, with reference to our $\sim^!_\kappa$ notation, $u\overset{a^\omega}{\Rightarrow}^!$ iff $u\overset{a^\omega}{\Rightarrow}$ and $\nexists u' . u\overset{a}{\Rightarrow} u'\overset{a^\omega}{\Rightarrow}$). Then, $u\overset{a}{\Rightarrow}_{d^\omega_a,0} u$, but $v\overset{a}{\Rightarrow}_{d^\omega_a,\delta} v' \implies \delta = \infty$.

**Definition 6.4.1 ($\mathbf{DD}'_\tau$)** *The* $d^\omega_{a_\tau}$-extended weak distances to disablings *functions are constructed: if $a \in \Sigma \cup \{\epsilon\}$, and $D$ is a finite (potentially empty) set of pairs $(d, \delta)$, where $d$ is $d^\omega_{a_\tau}$ or an already constructed $\mathrm{DD}'_\tau$ function, and $\delta \in \mathbb{N}_{-2,-1,\infty}$, then*

$$dd_\tau(a, D) = \lambda u . \min\{dist_\tau(u, v) \mid v\overset{a}{\not\Rightarrow}_D\}$$

*is a $\mathrm{DD}'_\tau$ function.* ■

**Lemma 6.4.1 ($\mathbf{DD}'_\tau$)**

*1. $u \approx v \implies u \equiv_{\mathrm{DD}'_\tau} v$; and*

*2. if $\approx = \equiv_{\mathrm{DD}'_\tau}$ on a class of processes, $\approx = \approx_{\omega^2}$ on that class.*

**Proof:** 1. If $d^\omega_{a_\tau}(u) \neq d^\omega_{a_\tau}(v)$ $I$ can force play to $u', v'$ with $u'\overset{a^\omega}{\Rightarrow} \iff v'\overset{a^\omega}{\not\Rightarrow}$, i.e. $\exists n . u' \not\approx_n v'$. The proof proceeds as per that of Lemma 6.3.3.2.

2. Let wlog $c_1 = d(u) < d(v)$. If $d = d^\omega_{a_\tau}$ then $I$ has a winning strategy on $G^\tau_{\omega+c_1}(u,v)$: force play (in $c_1$ moves) to $u', v'$ s.t. $d(u') = 0 < d(v')$, then there exists an $n$ s.t. $v' \overset{a}{\Rightarrow}^n$ but $u' \overset{a}{\not\Rightarrow}^n$. Otherwise, proceed as to the proof of Lemma 6.2.2. □

**Corollary 6.4.1 ($\mathrm{DD}'_\tau$ and BPA)** *On BPA processes,* $\mathrm{DD}'_\tau \subsetneq \approx$ *(see Equation 5.42).*

## 6.5 $\mathrm{DD}_\tau^{\mathcal{C},\mathcal{O}}$

Reaching further, we ask how many times a process is capable of performing a $\overset{a}{\Rightarrow}_D$ transition:

$$dd^\kappa_\tau(a,D)(u) =_{\mathrm{def}} \min\{\mathrm{dist}_\tau(u,v) \mid v \overset{a}{\not\Rightarrow}^\kappa_D\} \tag{6.34}$$

**Example 6.3** ($dd^\omega_\tau$)

1. *If $d = dd^\omega_\tau(a,D)$, then $d(u) > 0$ implies that, for every $n$, there exists a sequence $u \overset{a}{\Rightarrow}_D u_1 \overset{a}{\Rightarrow}_D u_2 \overset{a}{\Rightarrow}_D \ldots \overset{a}{\Rightarrow}_D u_n$.*

2. $d^\omega_{a_\tau} = dd^\omega_\tau(a,\emptyset)$

**Definition 6.5.1 ($\mathrm{DD}_\tau^{\mathcal{C},\mathcal{O}}$)** *The* $\mathcal{O}$-extended weak distances to disablings *functions are constructed: if $a \in \Sigma \cup \{\epsilon\}$, and $D$ is a (potentially empty) set of pairs $(d,\delta)$, where $d$ is an already constructed* $\mathrm{DD}_\tau^{\mathcal{C},\mathcal{O}}$ *function, $\delta \in \mathbb{N}_{-2,-1,\infty}$, and $0 > \kappa \in \mathcal{O}$, then*

$$dd^\kappa_\tau(a,D)$$

*is a* $\mathrm{DD}^{\mathcal{O}}_\tau$ *function.*[5] ■

The $\mathrm{DD}_\tau^{\mathcal{C},\kappa}$ functions put an upper limit of $\kappa$ on the ordinal in the functions' construction; the $\mathrm{DD}_\tau^{\aleph,\kappa}$ functions restrict both that and the size of the $D$-sets. In that, $\mathrm{DD}_\tau = \mathrm{DD}_\tau^{\aleph_0,1}$. Again, $\mathrm{DD}^\kappa_\tau$ abbreviates $\mathrm{DD}_\tau^{\aleph_0,\kappa}$

**Example 6.4** *The processes $u_\kappa, v_\kappa$ given in the proof of Lemma 6.2.1.1 (page 80) are distinguished by a* $\mathrm{DD}^{\mathcal{O}}_\tau$ *function of height 2, $dd_\tau(a, dd^\kappa_\tau(a,\emptyset),0)$. That is,*

$$v_\kappa \overset{a}{\Rightarrow}_{dd^\kappa_\tau(a,\emptyset),0} v_\kappa \tag{6.35}$$

*whereas,* $u_\kappa \overset{a}{\Rightarrow}_{dd^\kappa_\tau(a,\emptyset),\delta} u_\mu \implies \delta = -2$.

[5]We require $\kappa > 0$ to allow $\mathrm{DD}_\tau = \mathrm{DD}^1_\tau$. Trivially, if $d = dd^0_\tau(a,D)$ then $d(u) = \infty$.

It should be clear that the $\mathrm{DD}_\tau^{\mathcal{C},\mathcal{O}}$ functions respect bisimilarity,

$$u \approx v \implies u \equiv_{\mathrm{DD}_\tau^{\mathcal{C},\mathcal{O}}} v \tag{6.36}$$

(– non-$\mathrm{DD}_\tau^{\mathcal{C},\mathcal{O}}$-equivalence presents $I$ with a winning strategy). We make one further extension:

$$dd_\tau^\infty(a,D)(u) =_{\text{def}} \min\{\mathrm{dist}(u,u') \mid u' \overset{a}{\Rightarrow}{}_D^\infty\} \tag{6.37}$$

i.e. $dd_\tau^\infty(u) > 0$ when, after every $\overset{a}{\Rightarrow}_D$ move, it is always possible to make another one. Easily, $dd_\tau(a,D)(u) = \infty \implies dd_\tau^\infty(a,D) = \infty$ (and the contraimplication does not hold). On general processes, $dd_\tau^\infty(u) > 0$ iff $\forall\kappa \in \mathcal{O}.dd_\tau^\kappa(a,D)(u) > 0$, and on sub-$\aleph$-branching processes (for a regular $\aleph$), $dd_\tau^\infty(a,D)(u) = dd_\tau^\aleph(a,D)(u)$ (see the proof of Lemma 6.3.3.3, page 86). In that case, we say that the $\mathrm{DD}_\tau^{\mathcal{C},\mathcal{O}}$ hierarchy has collapsed by level $\aleph$.

**Example 6.5 ($\mathbf{DD}_\tau^{\mathcal{C},\mathcal{O}}$ on weakly-finitely branching processes)** *On processes $u$ whose branching modulo $\overset{a}{\Rightarrow}$ is finite (the silently lossy BPP of §5.2, say),*

$$dd_\tau^\omega(a,D) = dd_\tau^\infty(a,D) \tag{6.38}$$

*Adding $dd_\tau^\kappa(a,D)$ functions as tests, for any $\kappa > \omega$, cannot increase our ability to distinguish non-weakly bisimilar processes.*

These functions' connection to approximant collapse is subtle; that $\approx\, = \,\equiv_{\mathrm{DD}_\tau^{\aleph,\mathcal{O}}}$ (for any $\aleph$) on a class of processes enables us to conclude nothing about its level of collapse. If $\approx\, = \,\equiv_{\mathrm{DD}_\tau^{\aleph,\kappa}}$ (for a regular $\aleph$) Player $II$ has fewer than $\aleph$ opportunities to postpone loss of the game for $\kappa + \omega$ moves; we conclude,

$$\approx\, = \,\approx_{(\kappa+\omega)\aleph} \tag{6.39}$$

# Chapter 7

# DD$_\mathcal{T}$ on BPP

Weak bisimilarity is known to be decidable on the *totally normed* and *purely-generated normed* subclasses of BPP. We know of no non-trivial decidability results for unnormed BPP processes; no general method which could, for example, be applied to the processes $P, Q$ of Figure 7.1.

$$\begin{tikzcd}
 & P \arrow[r, "\tau"] \arrow[d, "\tau"] & A \arrow[loop right, "a"] & & \\
\epsilon & Q \arrow[l, "a"] \arrow[r, bend left, "\tau"] & QQ \arrow[l, "a"] \arrow[r, bend left, "\tau"] & QQQ \arrow[l, "a"] \arrow[r, bend left, "\tau"] & \cdots \arrow[l, "a"]
\end{tikzcd}$$

$$P \xrightarrow{\tau} A, P \xrightarrow{\tau} Q$$
$$Q \xrightarrow{\tau} QQ, Q \xrightarrow{a} \epsilon$$
$$A \xrightarrow{a} A$$

Figure 7.1: $P \approx^!_\omega Q$

But consider, $P$ can move to an unnormed state $A$, while any move from $Q$ must reach a state $Q^n$ of finite norm. In terms of the basic DD$_\mathcal{T}$ function $d_{a_\tau} = dd_\mathcal{T}(a, \emptyset)$,

$$P \overset{\epsilon}{\Rightarrow}_{d_{a_\tau},\infty} A \text{ and } Q \overset{\epsilon}{\not\Rightarrow}_{d_{a_\tau},\infty} \tag{7.1}$$

$(Q \overset{\epsilon}{\Rightarrow}_{d_{a_\tau},\delta} Q^{n+1} \implies \delta = n+2,\ Q \overset{\epsilon}{\Rightarrow}_{d_{a_\tau},-2} E)$; the DD$_\mathcal{T}$ function $d = dd(\epsilon, d_{a_\tau}, \infty)$ distinguishes them:

$$d(P) = 1 > d(Q) = 0 \tag{7.2}$$

## 7.1 Difficulties

The DD functions behave particularly well on BPP processes, in stark contrast to DD$_\mathcal{T}$. For example, for any $d \in$ DD,

$$\forall \alpha, \beta . d(\alpha\beta) = d(\alpha) + d(\beta) \tag{7.3}$$

(By Lemma 5.3.4, for each $d \in$ DD we can find a set of places $Q$ s.t. $d =$

$\textsc{norm}(Q)$. Of course, $\textsc{norm}(Q)(\alpha\beta) = \textsc{norm}(Q)(\alpha) + \textsc{norm}(Q)(\beta)$.) This is easily seen to fail for $\mathrm{DD}_\tau$, Figure 7.2.

$$\begin{tikzcd}
XX \arrow[r, "\tau"] \arrow[d, "\tau"] & XA \arrow[r, "\tau"] \arrow[dl, "a"] & AA \arrow[dl, "a"] \\
X \arrow[r, "\tau"] \arrow[d, "\tau"] & A \arrow[dl, "a"] & \\
{} & &
\end{tikzcd} \qquad d = dd_\tau(\epsilon, d_a, 2)$$

Figure 7.2: $d(XX) = 1 > d(X) + d(X) = 0$

then, $d = dd_\tau(\epsilon, \{(d_a, 2)\})$, and $d(XX) = 1 > d(X) + d(X) = 0$. The problem of finding nice algebraic properties for $\mathrm{DD}_\tau$ functions has proven very challenging.

**Example 7.1 (properties of $d_{a_\tau}$)** *For all processes $\alpha$, $d_{\epsilon_\tau}(\alpha) = \infty$.*

$$d_{a_\tau}(\alpha\beta) = \begin{cases} d_{a_\tau}(\alpha) + d_{a_\tau}(\beta) & \textit{if } d_{a_\tau}(\alpha) = 0 \textit{ and } d_{a_\tau}(\beta) = 0 \\ & \textit{or } d_{a_\tau}(\alpha) > 1 \textit{ and } d_{a_\tau}(\beta) > 1 \\ d_{a_\tau}(\alpha) + d_{a_\tau}(\beta) - 1 & \textit{otherwise} \end{cases} \tag{7.4}$$

## 7.2 Computability of the $\mathrm{DD}_\tau$ functions

In §5.5 we defined the $\mathrm{Reach}_\tau$ relation, Equation 5.25 (page 72), used to prove the semilinearity of $\overset{a}{\Rightarrow}$ (and from that §5.6, the decidability of the finite approximants, $\approx_n$). We will proceed to use it to define Presburger formula for $\mathrm{dist}_\tau$, and develop from that formulas for the (finite) $\mathrm{DD}_\tau$ functions, in order to prove decidability.

Denote the Presburger equivalent of $\mathrm{Reach}_\tau$ by $\psi$. That is, for all BPP processes $\alpha, \beta$,

$$\psi[\vec{\alpha}, \vec{\sigma}, \vec{\beta}] \quad \Longleftrightarrow \quad \mathrm{Reach}_\tau(\vec{\alpha}, \vec{\sigma}, \vec{\beta}) \tag{7.5}$$

Define,

$$\Delta(\vec{\alpha}, n, \vec{\beta}) \equiv \begin{array}{l} n = 0 \wedge \vec{\alpha} = \vec{\beta} \quad \text{or} \\ \exists \vec{\sigma}. \sum \vec{\sigma} = n \mathbin{\dot{-}} 1 \wedge \psi(\vec{\alpha}, \vec{\sigma}, \vec{\alpha}) \end{array} \tag{7.6}$$

$$\Delta^{!}(\vec{\alpha}, n, \vec{\beta}) \equiv \Delta(\vec{\alpha}, n, \vec{\beta}) \wedge \not\exists n' < n. \Delta(\vec{\alpha}, n', \vec{\beta}) \tag{7.7}$$

(Where $n \mathbin{\dot{-}} m = 0$ if $n - m < 0$, and $n - m$ otherwise.) It is straightforward to verify that:

**Lemma 7.2.1 ($\Delta^{!}$)**

1. *$\mathit{dist}_\tau(\alpha, \beta) = n$ iff $\Delta^{!}[\vec{\alpha}, n, \vec{\beta}]$; and*

2. *$dist_\tau(\alpha,\beta) = \infty$ iff $\not\exists n.\Delta^![\vec{\alpha}, n, \vec{\beta}]$*

If $\phi$ is a Presburger Formula which expresses a set of processes, we define the distance until it is reached,

$$\Delta_\phi(\vec{\alpha}, n) \equiv \exists\vec{\beta}.\phi(\vec{\beta}) \wedge \Delta(\vec{\alpha}, n, \vec{\beta}) \tag{7.8}$$

$$\Delta^!_\phi(\vec{\alpha}, n) \equiv \Delta_\phi(\vec{\alpha}, n) \wedge \not\exists n' < n.\Delta_\phi(\vec{\alpha}, n') \tag{7.9}$$

For $V = \{X_1, \ldots, X_k\}$, let $\{X_{n_1}, \ldots, X_{n_l}\} = \{X \in V \mid X\overset{a}{\Rightarrow}\}$, and define $\phi_a(\vec{x}) \equiv \forall i.0 < i \leq l.\vec{x}_{n_i} = 0$ (i.e. $\phi_a(\vec{\alpha}) \iff \alpha\overset{a}{\not\Rightarrow}$), then

$$D_a(\vec{x}, n) =_{\text{def}} \Delta^!_{\phi_a}(\vec{x}, n) \tag{7.10}$$

and we find,

**Lemma 7.2.2** ($D_a$)

1. *$dd(a,\emptyset)(\alpha) = n$ iff $D_a[\vec{\alpha}, n]$; and*

2. *$dd(a,\emptyset)(\alpha) = \infty$ iff $\not\exists n.D_a[\vec{\alpha}, n]$*

Consider some finite DD function $dd(a, D)$ (that is, $dd(a, D) \in \mathrm{DD}^{\aleph_0}$), and imagine that for each $(d_i, \delta_i) \in D$ we have a Presburger formula $D_i$ with, $d_i(\alpha) = n$ iff $D_i[\vec{\alpha}, n]$ and $d_i(\alpha) = \infty$ iff $\not\exists n.D_i[\vec{\alpha}, n]$. Define,

$$D^\infty = \{d \mid (d, \infty) \in D\} \tag{7.11}$$

$$D^n = D - D^\infty \tag{7.12}$$

We wish to construct a Presburger formula $\phi$ s.t. $\phi[\vec{\alpha}, n]$ iff $\alpha\overset{a}{\not\Rightarrow}_D$:

$$\begin{aligned}
\psi_{a,D}(\vec{\alpha}) \equiv & \bigvee_{(d_i,\delta_i)\in D} \not\exists n.D_i(\vec{\alpha}, n) \quad \vee & (7.13)\\
& \forall\vec{y}.\phi(\vec{x}, a, \vec{y}) \implies & (7.14)\\
& \quad \bigvee_{d_i\in D^\infty} \exists n.D_i(\vec{y}, n) \quad \vee & (7.15)\\
& \quad \bigvee_{(d_i,\delta_i)\in D^n} \not\exists n.D_i(\vec{x}, n) \wedge D_i(\vec{y}, n + \delta_i) & (7.16)
\end{aligned}$$

Where $\alpha\overset{a}{\not\Rightarrow}_D$ if and only if:

1. There exists a $(d, \delta) \in D$ s.t. $d(\alpha) = \infty$ (Line 7.13); or

2. For all $\beta.\alpha\overset{a}{\Rightarrow}\beta$ (Line 7.14)

(a) $(d, \infty) \in D$ yet $d(\beta) = n < \infty$ (Line 7.15); or

(b) $(d, \delta) \in D$ yet $d(\alpha) + \delta \neq d(\beta)$ (Line 7.16)

Specifically, $\phi_{a,\emptyset}(\vec{x}) \equiv \forall \vec{y}. \neg\psi(\vec{x}, a, \vec{y})$. We have,

$$DD_{a,D} = \Delta^!_{\phi_{a,D}} \tag{7.17}$$

**Lemma 7.2.3 ($DD_{a,D}$)**

*1. $dd(a, D)(\alpha) = n$ iff $DD_{a,D}[\vec{\alpha}, n]$; and*

*2. $dd(a, D)(\alpha) = \infty$ iff $\nexists n. DD_{a,D}[\vec{\alpha}, n]$*

And as a corollary,

**Theorem 7.2.1 ($\mathbf{DD}_\tau$ functions on BPP)** *The $\mathrm{DD}_\tau$ functions are computable on BPP processes.*

**Proof:** Theorem 2.4.2. To compute, say, $d = dd(a, D)$ on $\alpha$, construct its Presburger equivalent $DD_{a,D}$, then if $\nexists n. DD_{a,D}[\vec{\alpha}, n]$ is false, proceed with

$$DD_{a,D}[\vec{\alpha}, 0], DD_{a,D}[\vec{\alpha}, 1], \ldots$$

until we find a formula that is satisfied. □

To compute $\equiv_d$, for some $d = dd(a, D) \in u\mathrm{DD}$,

$$\alpha \equiv_d \beta \quad \text{iff} \quad \forall n. DD_{a,D}[\alpha, n] \iff DD_{a,D}[\beta, n] \tag{7.18}$$

### 7.2.1 Finite approximability

If $\alpha \not\approx_\omega \beta$, Lemma 6.2.3 (Example 6.2) and Theorem 7.2.1 imply that we can find a function $d \in \mathrm{DD}_\tau$ such that $d(\alpha) \neq d(\beta)$. In this way we add a third method for semideciding $\not\approx_\omega$ on BPP processes – and deciding $\approx$ on the finitely approximable BPP – to the approximants of Theorem 5.6.1, and the strong bisimulation games of Corollary 5.6.1.

**Example 7.2 (trace and failures equivalence)**

*Weak trace and weak failures non-equivalences (Equation 4.16) are semidecidable on BPP processes through the $\mathrm{DD}_\tau$ functions. That is, if $\alpha \not\equiv_L \beta$ or $\alpha \not\equiv_f \beta$ then $\exists d \in \mathrm{DD}_\tau \,.\, d(\alpha) \neq d(\beta)$.*

### 7.2.2 Unary, non-norm-zero BPP

In [Stř99], Stříbrná proves that on BPPs with a single action name and no variables of zero norm, $\approx\ =\ \approx_{\omega\cdot 2}$. We find:

**Lemma 7.2.4 (unary, non-zero-norm)** *On BPP processes with a single action name and no transitions of the form $X\xrightarrow{\tau}\epsilon$, $\approx\ =\ \equiv_{\mathrm{DD}_\tau}$ (and so $\approx$ is decidable).*

**Proof:** Let $\alpha \not\approx_\kappa \beta$, an induction on $\kappa$. If $\alpha \not\approx_1 \beta$ then $d_{a_\tau}(\alpha) = 0 \iff d_{a_\tau}(\beta) \neq 0$. For the inductive step, let $I : \alpha \overset{a}{\Rightarrow} \alpha'$, and observe first that:

$$\mathrm{norm}_\tau(\alpha) = \infty = \mathrm{norm}_\tau(\beta) \implies \alpha \approx \beta \tag{7.19}$$

If either $\mathrm{norm}_\tau(\alpha) = \infty$ or $\mathrm{norm}_\tau(\alpha) = \infty$ then $d_{a_\tau}$ distinguishes them. If not, observe second that if $d_{a_\tau}(\alpha') < \infty$,

$$B = \{\beta' \mid \beta \overset{a}{\Rightarrow} \beta' \text{ and } d_{a_\tau}(\beta') = d_{a_\tau}(\alpha')\} \tag{7.20}$$

is a finite set (false for the full unary BPP). By hypothesis for each $\beta' \in B$ there exists a $d_{\beta'} \in \mathrm{DD}_\tau$ with $d_{\beta'}(\beta') \neq d_{\beta'}(\alpha')$. Our $\mathrm{DD}_\tau$ function is then:

$$d = dd_\tau(a, D) \quad \pi_1(D) = (d_{a_\tau}) \cup (d_{\beta'})_{\beta' \in B} \quad \alpha \overset{a}{\Rightarrow}_D \alpha' \tag{7.21}$$

That is, $D$ is a set of $\mathrm{DD}_\tau \times \mathbb{N}_{-2,-1,\infty}$ pairs, whose $\mathrm{DD}_\tau$ parts equal $\{d_{a_\tau}\} \cup \{d_{\beta'} \mid \beta' \in B\}$, and whose $\delta$ parts are such that $\alpha \overset{a}{\Rightarrow}_D \alpha'$ is *true*. □

Whether having a single-letter alphabet is sufficient in itself to give $\approx\ =\ \equiv_{\mathrm{DD}_\tau}$ is listed as Open Problem 1.3 in Chapter 8. I have been unable to find a counter-example.

## 7.3 Insufficiency

Figure 6.3 (§6.4, page 88) gives a pair of processes $u, v$ which are related by $\mathrm{DD}_\tau$ but are not weakly bisimilar. It should be clear that while we can find a BPP process which is equivalent to $u$,

$$X \xrightarrow{\tau} XA, \epsilon \; A \xrightarrow{a,\tau} \epsilon \qquad X \approx u \tag{7.22}$$

no $\beta$ weakly bisimilar to $v$ exists, as the ability to perform an arbitrary sequence of $a$ actions implies that a process is able to make infinitely many (Lemma 7.5.1). And yet, an analogue does exist, Figure 7.3.

$$X \xrightarrow{\tau} XA \quad X \xrightarrow{c} X \quad X \xrightarrow{\tau} C$$
$$A \xrightarrow{a,\tau} \epsilon$$
$$Y \xrightarrow{\tau} YA \quad Y \xrightarrow{\tau} C$$
$$C \xrightarrow{c} C \quad C \xrightarrow{e} \epsilon$$

$$\begin{tikzcd}
X \arrow[loop above, "c"] \arrow[r, bend left, "\tau"] \arrow[d, "\tau"] & XA \arrow[loop above, "c"] \arrow[l, "{a,\tau}"] \arrow[r, bend left, "\tau"] \arrow[d, "\tau"] & XAA \arrow[loop above, "c"] \arrow[l, "{a,\tau}"] \arrow[r, bend left, "\tau"] \arrow[d, "\tau"] & \cdots \arrow[l, "{a,\tau}"] \\
C \arrow[loop left, "c"] \arrow[d, "e"] & CA \arrow[l, "{\tau,d}"] \arrow[loop left, "c"] \arrow[d, "e"] & CAA \arrow[l, "{a,\tau}"] \arrow[loop left, "c"] \arrow[d, "e"] & \cdots \arrow[l, "{a,\tau}"] \\
\tau & A \arrow[l, "{a,\tau}"] & AA \arrow[l, "{a,\tau}"] & \cdots \arrow[l, "{a,\tau}"]
\end{tikzcd}
\qquad
\begin{tikzcd}
Y \arrow[r, bend left, "\tau"] \arrow[d, "\tau"] & YA \arrow[l, "{a,\tau}"] \arrow[r, bend left, "\tau"] \arrow[d, "\tau"] & YAA \arrow[l, "{a,\tau}"] \arrow[r, bend left, "\tau"] \arrow[d, "\tau"] & \cdots \arrow[l, "{a,\tau}"] \\
C \arrow[loop left, "c"] \arrow[d, "e"] & CA \arrow[l, "{a,\tau}"] \arrow[loop left, "c"] \arrow[d, "e"] & CAA \arrow[l, "{a,\tau}"] \arrow[loop left, "c"] \arrow[d, "e"] & \cdots \arrow[l, "{a,\tau}"] \\
\tau & A \arrow[l, "{a,\tau}"] & AA \arrow[l, "{a,\tau}"] & \cdots \arrow[l, "{a,\tau}"]
\end{tikzcd}$$

Figure 7.3: $X \equiv_{\mathrm{DD}_\tau} Y$ but $X \not\approx^!_\omega Y$

The proof that $X \equiv_{\mathrm{DD}_\tau} Y$ proceeds in a similar (though more tedious) manner to that of §6.4 (page 88), so we will not cover it in depth here. The only subtle point to check is the move which guarantees non-weak bisimilarity: $X$ can make an $\approx$-neutral transition on a $c$ action, $X \overset{c}{\Rightarrow} X$, while if $Y \overset{c}{\Rightarrow} \beta$, $Y \not\approx \beta$. But again, for any $d \in \mathrm{DD}_\tau$ we can find an $n$ such that for all $i \geq n$, $d(Y) = d(CA^i)$, so on any candidate $d = dd(c, D)$, with $X \overset{c}{\Rightarrow}_D X$, $\exists i. Y \overset{c}{\Rightarrow}_D CA^i$.

**Lemma 7.3.1 ($\mathbf{DD}_\tau$ and normed processes)** *On normed BPP,* $\approx \subsetneq \equiv_{\mathrm{DD}_\tau}$.

It is worth noting that $X$ and $Y$ are normed but are not *purely generated* (§5.4.2). $G(X) = \{A\} = G(Y)$, yet $X, Y \xrightarrow{\epsilon}_{\mathrm{norm},0} C$ (i.e. both can make non-generating, norm-neutral moves). Open Problem 1 asks whether there exists a normed and purely generated pair of processes whose inequivalence cannot be told by $\mathrm{DD}_\tau$.

Casting ones mind back to the van Glabbeek hierarchy Figure 1.2, every equivalence coarser than bisimilarity is undecidable. Analogously, $\equiv_{\mathrm{DD}_\tau}$ is coarser than weak bisimilarity, and finer than weak trace equivalence (which is certainly undecidable); it would be extraordinary if the finite weak distances-to-disablings were decidable on general BPP.

**Conjecture 7.3.1 ($\mathbf{DD}_\tau$ on BPP)** $\equiv_{\mathrm{DD}_\tau}$ *is undecidable on BPP processes.*

## 7.4 $\mathrm{DD}'_\tau$

The $\mathrm{DD}'_\tau$ functions (Definition 6.5.1, page 90) extend $\mathrm{DD}_\tau$ with $d^\omega_{a_\tau}$ functions. They are strong enough to distinguish $X, Y$ of Figure 7.3,

$$d = dd(c, d^\omega_{d_\tau}, 0) \qquad d(X) = 1 > d(Y) = 0 \tag{7.23}$$

and is easily seen to preserve computability. For this, it suffices to prove that we can effectively construct a formula $\Omega_a$ of Presburger Arithmetic such that,

$$\Omega_a[\vec{\alpha}] \iff \alpha \overset{a^\omega}{\Rightarrow} \tag{7.24}$$

Let our variables be $V = \{X_1, \ldots, X_k\}$. Define,

$$\begin{aligned} A(X) &= \{Y \mid X \overset{\underline{a}}{\Rightarrow} \alpha Y\} & (7.25)\\ A^{n+1}(X) &= \{Y \mid Z \in A^n(X), Z \overset{\underline{a}}{\Rightarrow} \alpha Y\} & (7.26) \end{aligned}$$

Then, $A^k(X) = A^{k+1}(X)$, $X \in A^k(X) \implies X \overset{a^\omega}{\Rightarrow}$, and

$$X \overset{a^\omega}{\Rightarrow} \text{ iff } \exists Y \in A^k(X). Y \in A^k(Y) \tag{7.27}$$

Let $A^\omega = \{X \mid \exists Y \in A^k(X). Y \in A^k(Y)\} = \{X_{x_1}, X_{x_2}, \ldots, X_{x_l}\}$. $\alpha \overset{a^\omega}{\Rightarrow}$ iff $\exists X \in A^\omega . \alpha(X) > 0$, hence define

$$\Omega(\vec{\alpha}) =_{\text{def}} \bigvee_{i=1}^{l} \vec{\alpha}(x_i) > 0 \tag{7.28}$$

**Lemma 7.4.1 ($\mathrm{DD}'_\tau$ on BPP)** *The* $\mathrm{DD}'_\tau$ *functions are computable on BPP processes.*

The $\mathrm{DD}'_\tau$ functions are able to express weak bisimilarity on every example BPP process-pair I have managed to find[1] yet a proof that that they capture the problem on full BPP has remained elusive, and is probably illusive:

**Conjecture 7.4.1 ($\approx \subsetneq \equiv_{\mathrm{DD}'_\tau}$)** *There exist BPP processes $\alpha, \beta$ with $\alpha \not\approx \beta$ yet $\alpha \equiv_{\mathrm{DD}'_\tau} \beta$.*

## 7.5 $\mathrm{DD}^{\mathcal{O}}_\tau$, and its collapse

**Lemma 7.5.1 ($d^\infty_a$)** *On BPP processes, $\alpha \overset{a^\omega}{\Rightarrow} \iff \alpha \overset{a^\infty}{\Rightarrow}$.*

[1]Conversely, it is obvious that $\approx \subsetneq \equiv_{DD'_\tau}$ on BPA processes – Lemma 6.2.2 would imply that $\approx = \approx_{\omega^2}$, but this is contradicted by the BPA processes of Equation 5.42.

**Proof:** A corollary of the construction used in Equation 7.24. $\square$

That is, we cannot strengthen $\equiv_{\mathrm{DD}'_\tau}$ by adding any function $dd^{\kappa}_{\tau}(a, \emptyset)$ (§6.5). Whether the same holds for the $dd^{\kappa}_{\tau}(a, D) \in \mathrm{DD}^{\mathcal{O}}_{\tau}$ functions depends on the truth of Conjecture 7.4.1. The final chapter constitutes a research programme to answer the question of how complex our distances-to-disablings functions need be to pin weak bisimilarity; this chapter closes with a conjecture.

**Conjecture 7.5.1 ($\mathbf{DD}^{\omega}_{\tau}$ on BPP)**

1. *Each* $\mathrm{DD}^{\omega}_{\tau}$ *function is computable;*
2. *and* $\approx \, = \, \equiv_{\mathrm{DD}^{\omega}_{\tau}}$.

The result, were part one of this conjecture proven, would be to take the known level of approximant collapse from $\omega^{\omega}$ to $\omega^2$; if in addition the second part is true, then $\approx$ is decidable on the Basic Parallel Processes.

# Chapter 8

# Conclusions, and a research programme

Bisimilarity – behavioural equivalence – is a well-understood problem on the Basic Parallel Processes, with a number of decidability proofs using several different techniques. Weak bisimilarity, in contrast, remains an open problem. While semidecidability was shown over a decade ago, the problem of semideciding *in*equivalence has resisted every attempt made at it, and is known to be true only for heavily restricted subclasses. The central difficulty is in moving from a finitely branching system (in which each step yields only a finite number of possibilities) to one that branches infinitely; semideciding inequivalence switches from being a trivial problem to a formidable one. Nevertheless, it is generally believed that weak bisimilarity is decidable, and it is its closeness to the edge of undecidability which makes it an interesting question to tackle.

This thesis makes two contributions to the theory of Basic Parallel Processes, both with a view to the semidecidability of inequivalence. The first concerns weak bisimulation approximant collapse, and the second is a development of Jančar's distances-to-disablings functions.

If two processes are not weakly bisimilar we can attach an ordinal number to the number of moves it takes to manifest this inequivalence. If our processes branch finitely this will always be a finite number; Basic Parallel Processes (with silent moves) branch infinitely, and it is not difficult (given any natural number $n$) to find processes whose inequivalence is not manifested until $\omega + n$. A long-standing conjecture holds that it is impossible to find Basic Parallel Processes whose inequivalence is not seen until $\omega \times 2$ steps, but before this thesis the only work on the problem has achieved the (rather trivial) result that no such processes exist which cannot be told by $\omega_1^{CK}$. Our work takes this number down to $\omega^\omega$ – the first "sensible" bound on the problem. We achieve this using a novel constructive version of Dickson's Lemma.

The major work of this thesis is a development of the distances-to-disablings functions, invented by Petr Jančar in order to express *strong* bisimilarity on the Basic Parallel Processes, into a tool applicable to weak bisimilarity. In this we have met with partial but encouraging success. The basic distances-to-disablings functions are computable on Basic Parallel Processes, and are able to distinguish inequivalent processes which the current methods cannot; but unfortunately they do not fully express weak bisimilarity – if two processes are not weakly bisimilar, we cannot guarantee that a distance-to-disablings function exists which will tell the difference. From there, we created a natural extension of the distances-to-disablings, one that retains computability, enabling them to handle any example I have yet been able to find. However, I have been unable to prove that this is enough to capture weak bisimilarity, and indeed now believe that a stronger (potentially uncomputable) extension is required.

We close the thesis with a research programme: four open problems to complete the theory of weak distances-to-disablings on the Basic Parallel Processes, and, we hope, finally settle weak bisimilarity there.

## 8.1 A programme

The $\mathrm{DD}_\tau$ – and $\mathrm{DD}'_\tau$ – functions are computable, and will distinguish processes for which the currently developed methods are inapplicable (Figure 7.1, page 92); but they do not express weak bisimilarity on full BPP (Figure 7.3, page 97), and we have not found a non-trivial restriction on BPP processes which yields $\approx \, = \, \equiv_{\mathrm{DD}_\tau}$.

**Open problem 1 ($\mathrm{DD}_\tau$)**

1. *Find a necessary and sufficient restriction on BPP processes to give* $\approx \, = \, \equiv_{\mathrm{DD}_\tau}$.

2. *Every example found thus far of non-weakly bisimilar,* $\mathrm{DD}_\tau$*-equivalent processes happens to not be purely generated. Is* normed, purely generated *a sufficient condition?*

3. *Being unary* and *having no variables of zero norm is sufficient, but is having a single observable action also sufficient?*

4. *Is* $\equiv_{\mathrm{DD}_\tau}$ *undecidable on full BPP?*

The extended functions, $\mathrm{DD}'_\tau$, retain computability, and we have not found an example of non-weakly bisimilar BPP processes which are nevertheless $\mathrm{DD}'_\tau$-equivalent;

a proof that no such example exists is sufficient to give the decidability of $\approx$ on BPP. However, we think that a counterexample does exist:

**Open problem 2 ($\mathbf{DD}'_\tau$)** *Find BPP processes $\alpha, \beta$ for which $\alpha \not\approx \beta$ but $\alpha \equiv_{\mathrm{DD}'_\tau} \beta$.*

Moving to the extended $\mathrm{DD}^{\mathcal{O}}_\tau$ functions, it is easily seen that $d^{\omega}_{a_\tau} = d^{\infty}_{a_\tau}$, but rather harder to prove that the hierarchy collapses at $\omega$ for more complex $\mathrm{DD}^{\mathcal{O}}_\tau$ functions:

**Open problem 3 ($\mathbf{DD}^{\omega\times 2}_\tau$)** *Can we find a $dd_\tau(a, D) \in \mathrm{DD}^{\mathcal{O}}_\tau$ and a BPP $\alpha$ such that $dd^{\omega}_\tau(a, D)(\alpha) \neq dd^{\omega+1}_\tau(a, D)(\alpha)$?*

Chapter 7 ends by conjecturing (7.5.1) that the $\mathrm{DD}^{\omega}_\tau$ functions are strong enough to express $\approx$ on full BPP processes. A corollary is to take the upper bound on approximant collapse from $\omega^\omega$ (§5.7) to $\omega^2$. If, like $\mathrm{DD}'_\tau$, each $\mathrm{DD}^{\omega}_\tau$ function is computable, then $\not\approx$ becomes semidecidable, and (in conjunction with Lemma 5.5.2) the long-open problem of the decidability of $\approx$ on Basic Parallel Processes will be closed, positively.

**Open problem 4 ($\mathbf{DD}^{\omega}_\tau$)**

1. *Does $\equiv_{\mathrm{DD}^{\omega}_\tau}$ equal $\approx$ on BPP processes; and*

2. *Are the $\mathrm{DD}^{\omega}_\tau$ functions computable?*

# Bibliography

[ÁBGS91] C Álvarez, J. L. Balcázar, J. Gabarró, and M. Sántha. Parallel complexity in the design and analysis of concurrent systems. In *PARLE: Parallel Architectures and Languages Europe*. LNCS, Springer-Verlag, 1991.

[Ace03] L. Aceto. Some of my favourite results in classic process algebra. *Bulletin of the EATCS*, 81:90–108, 2003.

[AFV01] L. Aceto, W. J. Fokkink, and C. Verhoef. Structural operational semantics. In Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors, *Handbook of Process Algebra, Chapter 3*, pages 197–292. Elsevier Science, Dordrecht, The Netherlands, 2001.

[Bas96] T. Basten. Branching Bisimilarity is an Equivalence indeed! *Information Processing Letters*, 58(3):141–147, May 1996.

[BBK87] J. C. M. Baeten, J. A. Bergstra, and J. W. Klop. Decidability of bisimulation equivalence for processes generating context-free languages. In A. J. Nijman J. W. de Bakker and P. C. Treleaven, editors, *Proceedings of the Conference on Parallel Architectures and Languages Europe (PARLE). Volume II: Parallel Languages*, volume 259 of *LNCS*, pages 94–111, Eindhoven, The Netherlands, June 1987. Springer-Verlag.

[BBK93] J. C. M. Baeten, J. A. Bergstra, and J. W. Klop. Decidability of bisimulation equivalence for processes generating context-free languages. *Journal of the ACM*, 40(3):653–682, July 1993.

[BCMS01] O. Burkart, D. Caucal, F. Moller, and B. Steffen. *Handbook of Process Algebra*, chapter 9, pages 545–623. Prentice-Hall, North-Holland, 2001. Ch: Verification on Infinite Structures.

[BE97] O. Burkart and J. Esparza. More infinite results. *Bulletin of the European Association for Theoretical Computer Science*, 62:138–159, June 1997. Columns: Concurrency.

[BE99] J. Barwise and J. Etchemendy. *Language, Proof, and Logic.* CSLI Publications, Stanford, California, 1999.

[BG06] A. Blass and Y. Gurevich. Program termination, and well partial orderings. Technical Report MSR-TR-2006-27, Microsoft Research (MSR), March 2006.

[BHR84] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31(3):560–599, July 1984.

[BK85] J. A. Bergstra and J. W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37:77–121, 1985.

[Bla95] J. Blanco. Normed BPP and BPA. In *Proceedings of the ACP'94*, pages 242–251. Springer-Verlag, 1995.

[BLM89] K. Buning, T. Lettmann, and E. Mayr. Projections of vector addition system reachability sets are semilinear. *Theoretical Computer Science*, 64, 1989.

[BM99] A. Bouajjani and R. Mayr. Model checking lossy vector addition systems. In *STACS: Annual Symposium on Theoretical Aspects of Computer Science*, volume 1563, page 323, 1999.

[Boo87] R. V. Book. Thue systems as rewriting systems. *J. Symb. Comput.*, 3(1-2):39–68, 1987.

[Bos96] D. Bosscher. Decidability of bisimulation equivalence of context-free processes extends to processes defined over BPA$\delta$. Technical report, Centrum voor Wiskunde en Informatica, 1996.

[BPS01] J. A. Bergstra, A. Ponse, and S. A. Smolka, editors. *Handbook of Process Algebra.* North-Holland, 2001.

[BS90] J. Bradfield and C. Stirling. Verifying temporal properties of processes. In J. C. M. Baeten and J. W. Klop, editors, *CONCUR '90: Theories of Concurrency: Unification and Extension*, volume 458 of *Lecture Notes in Computer Science*, pages 115–125, Amsterdam, The Netherlands, 27–30August 1990. Springer-Verlag.

[But72] P. Butzbach. Une famille de congruences de thue pour lesquelles le problème de l'équivalence est décidable. application á l'équivalence des grammaires séparées. In *ICALP*, pages 3–12, 1972.

[BvG87] J. C. M. Baeten and R. J. van Glabbeek. Another look at abstraction in process algebra. In *ICALP: Annual International Colloquium on Automata, Languages and Programming*, 1987.

[Cau90] D. Caucal. Graphes canoniques de graphes algébriques. *Theoretical Informatics and Applications*, 24(4):339–352, 1990.

[Cau92] D. Caucal. On the regular structure of prefix rewriting. *Theoretical Computer Science*, 106(1):61–86, November 1992.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–265, April 1986.

[CHM93] S. Christensen, Y. Hirshfeld, and F. Moller. Bisimulation equivalence is decidable for basic parallel processes. In *CONCUR*, pages 143–157, 1993.

[Cho56] N. Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 2(3):113–124, 1956.

[Cho57] N. Chomsky. *Syntactic structures.* The Hague: Mouton, 1957.

[Chr93] S. Christensen. *Decidability and Decomposition in Process Algebras.* PhD thesis, The University of Edinburgh, 1993. Also available as The University of Edinburgh, LFCS report ECS-LFCS-93-278.

[CHS92] S. Christensen, H. Hüttel, and C. Stirling. Bisimulation equivalence is decidable for all context-free processes. In W. R. Cleaveland, editor, *CONCUR '92: Third International Conference on Concurrency Theory*, volume 630 of *Lecture Notes in Computer Science*, pages 138–147, Stony Brook, New York, 24–27August 1992. Springer-Verlag.

[CHS95] S. Christensen, H. Hüttel, and C. Stirling. Bisimulation equivalence is decidable for all context-free processes. *Information and Computation*, 121(2):143–148, September 1995.

[CHT95] D. Caucal, T. Huynh, and L. Tian. Deciding branching bisimilarity of normed context-free processes is in $\Sigma_2^p$. *INFCTRL: Information and Computation (formerly Information and Control)*, 118, 1995.

[ČKK96] I. Černá, M. Kretínský, and A. Kucera. Bisimilarity is decidable in the union of normed BPA and normed BPP processes. *Electronic Notes Theoretical Computer Science*, 5, 1996.

[CM90] D. Caucal and R. Monfort. On the transition graphs of automata and grammars. In *WG: Graph-Theoretic Concepts in Computer Science, International Workshop WG*, 1990.

[Coh70] R. S. Cohen. Star height of certain families of regular events. *Journal of Computer and System Sciences*, 4(3):281–297, June 1970.

[CRM74] S. Crespi-Reghizzi and D. Mandrioli. Petri Nets and commutative grammars. Technical Report 74-5, Laboratorio di Calcolatori, Instituto di Electtrotecnia del Politecnico di Milano, 1974.

[dBZ82a] J. W. de Bakker and J. I. Zucker. Denotational semantics of concurrency. In *Proceedings of the 14th ACM Symposium on the Theory of Computing*, 1982.

[dBZ82b] J. W. de Bakker and J. I. Zucker. Processes and the denotational semantics of concurrency. *INFCTRL: Information and Computation (formerly Information and Control)*, 54, 1982.

[DGHP99] M. D'Agostino, D. M. Gabbay, R. Hänle, and J. Posegga, editors. *Handbook of Tableau Methods.* Kluwer Academic Publishers, Dordrecht, 1999.

[Dic13] L. E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with $n$ distinct prime factors. *Amer. J. Math.*, 35:413–422, 1913.

[DJ90] N. Dershowitz and J-P. Jouannaud. Rewriting systems. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B*, pages 244–320. Elsevier Science, 1990.

[Emm88] W. Emmerich. Timed and stochastic Petri Nets, October 1988.

[EN94] J. Esparza and M. Nielsen. Decidability issues for Petri Nets – a survey. *Bulletin of the EATCS*, 52:244–262, 1994.

[End77] H. B. Enderton. *Elements of Set Theory.* Academic Press, 1977.

[ES69] S. Eilenberg and M. P. Schützenberger. Rational sets in commutative monoids. *Journal of Algebra*, 13:173–191, 1969.

[Esp97] J. Esparza. Petri Nets, commutative context-free grammars, and basic parallel processes. *FUNDINF: Fundamenta Informatica*, 31, 1997.

[Esp01] J. Esparza. Grammars as processes. *Lecture Notes in Computer Science*, 2300:277–297, 2001.

[Fit96] M. C. Fitting. *First-Order Logic and Automated Theorem Proving.* Springer, New York, second edition, 1996.

[FR74] M. J. Fischer and M. O. Rabin. Super-exponential complexity of presburger arithmetic. Project MAC Tech. Memorandum 43, MIT, Cambridge, 1974.

[Fre68] P. Freyd. Redei's finiteness theorem for commutative semigroups. *Proc. Amer. Math. Soc.*, 19:1003, 1968.

[GH94] J. F. Groote and H. Hüttel. Undecidable equivalences for basic process algebra. *Information and Computation*, 115(2):354–371, December 1994.

[Gin66] S. Ginsburg. *The Mathematical Theory of Context-Free Languages.* McGraw-Hill, New York, 1966.

[Gir89] J-Y. Girard. *Proofs and types*, volume 7 of *Cambridge tracts in theoretical computer science.* Cambridge University Press, 1989.

[Gre65] S. A. Greibach. A new normal form theorem for context-free phrase structure grammars. *Journal of the ACM*, 12:42–52, 1965.

[Gre68] S. A. Greibach. A note on undecidable properties of formal languages. *Mathematical Systems Theory*, 2(1):1–6, 1968.

[Gro92] J. F. Groote. A short proof of the decidability of bisimulation for normed BPA-processes. *Information Processing Letters*, 42(3):167–171, May 1992.

[GS66] S. Ginsburg and E. H. Spanier. Semigroups, presburger formulas, and languages. *Pacific Journal of Mathematics*, 16(2):285–296, 1966.

[GS97] F. Gécseg and M. Steinby. *Tree Languages*, volume 3, Beyond words, chapter 6, page 1. Springer-Verlag, Berlin, 1997. Ch: Tree Languages.

[GW89] R. J. van Glabbeek and W. P. Weijland. Branching time and abstraction in bisimulation semantics (extended abstract). In *IFIP Congress*, pages 613–618, 1989.

[GW99] I. P. Gent and T. Walsh. Beyond NP: the QSAT phase transition. In *AAAI/IAAI*, pages 648–653, 1999.

[Hir93] Y. Hirshfeld. Petri Nets and the equivalence problem. In *CSL: 7th Workshop on Computer Science Logic.* LNCS, Springer-Verlag, 1993.

[Hir94a] Y. Hirshfeld. Congruences in commutative semigroups. Technical Report ECS-LFCS-94-291, Laboratory for Foundations of Computer Science, The University of Edinburgh, 1994.

[Hir94b] Y. Hirshfeld. Petri Nets and the equivalence problem. *Lecture Notes in Computer Science*, 832:165–174, 1994.

[Hir94c] Y. Hirshfield. Deciding equivalences in simple process algebras. Technical Report ECS-LFCS-94-294, The University of Edinburgh, 1994.

[Hir96] Y. Hirshfeld. Bisimulation trees and the decidability of weak bisimulations. *Electronic Notes Theoretical Computer Science*, 5, 1996.

[HJ99] Y. Hirshfeld and M. Jerrum. Bisimulation equivalence is decidable for normed process algebra (extended abstract). *Lecture Notes in Computer Science*, 1644:72–73, 1999.

[HJM96a] Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial algorithm for deciding bisimilarity of normed context-free processes. *Theoretical Computer Science*, 158(1–2):143–159, 20 May 1996.

[HJM96b] Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial-time algorithm for deciding bisimulation equivalence of normed Basic Parallel Processes. *Mathematical Structures in Computer Science*, 6(3):251–259, June 1996.

[HM05] W. J. Harwood and F. Moller. Weak bisimulation approximants. In *Selected Papers from the CALCO Young Researchers Workshop (CALCO-jnr 2005), Swansea Uni- versity Research Report CSR 18-2005*, pages 27–40, 2005.

[HMS06] W. J. Harwood, F. Moller, and A. Setzer. Weak bisimulation approximants. In Zoltán Ésik, editor, *CSL*, volume 4207 of *Lecture Notes in Computer Science*, pages 365–379. Springer-Verlag, 2006.

[Hoa78a] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, August 1978. See corrigendum [Hoa78b].

[Hoa78b] C. A. R. Hoare. Corrigendum: "Communicating Sequential Processes". *Communications of the ACM*, 21(11):958–958, November 1978. See [Hoa78a].

[HP79] J. E. Hopcroft and J-J. Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theoretical Computer Science*, 8:135–159, 1979.

[HR04] M. Huth and M. Ryan. *Logic in Computer Science.* Cambridge, second edition, 2004.

[HU87] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*, chapter 2, pages 13–45. Addison-Wesley Publishing Company, 1987.

[Hüt91a] H. Hüttel. *Decidability, Behavioural Equivalences and Infinite Transition Graphs.* Ph.D. thesis, Computer Science Dept., University of Edinburgh, December 1991.

[Hüt91b] H. Hüttel. Silence is golden: Branching bisimilarity is decidable for context–free processes. *Lecture Notes in Computer Science*, 575:2, 1991.

[Ing67] P. Z. Ingerman. "Panini-Backus Form" suggested. *Commun. ACM*, 10(3):137, 1967.

[Jan95a] P. Jančar. High undecidability of weak bisimilarity for Petri Nets. *Lecture Notes in Computer Science*, 915:349, 1995.

[Jan95b] P. Jančar. Undecidability of bisimilarity for Petri Nets and some related problems. *Theoretical Computer Science*, 148(2):281–301, 1995.

[Jan03] P. Jančar. Strong bisimilarity on basic parallel processes is PSPACE-complete. In *Proceedings of the eighteenth Annual IEEE Syposium on*

*Logic in Computer Science (LICS-03)9*, pages 218–227, Los Alamitos, CA, June 22–25 2003. IEEE Computer Society.

[JJ79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[JJ85] J. A. Bergstra and J. W. Klop. Algebra of Communicating Processes with Abstraction. *Theoretical Computer Science*, 37(1):77–121, 1985.

[JK04] P. Jančar and M. Kot. Bisimilarity on normed Basic Parallel Processes can be decided in time $O(n^3)$. 2004.

[JKM03] P. Jančar, A. Kučera, and F. Moller. Deciding bisimilarity between BPA and BPP processes. In *CONCUR: 14th International Conference on Concurrency Theory.* LNCS, Springer-Verlag, 2003.

[JKS05] P. Jančar, M. Kot, and Z. Sawa. Notes on complexity of bisimilarity between BPA and BPP. Work in progress, July 2005.

[JM99] P. Jančar and F. Moller. Techniques for decidability and undecidability of bisimilarity. In *CONCUR: 10th International Conference on Concurrency Theory.* LNCS, Springer-Verlag, 1999.

[Joh75] S. C. Johnson. Yacc: Yet another compiler compiler. Computer Science Technical Report #32, Bell Laboratories, Murray Hill, NJ, 1975.

[Kak87] S. C. Kak. The paninian approach to natural language processing. *Int. J. Approx. Reasoning*, 1(1):117–130, 1987.

[Kec95] A. S. Kechris. *Classical descriptive set theory.* Graduate texts in Mathematics. Springer-Verlag, 1995.

[KH66] A. J. Korenjak and J. E. Hopcroft. Simple deterministic languages. In *FOCS: IEEE Symposium on Foundations of Computer Science (FOCS)*, 1966.

[Klo90] J. W. Klop. Term rewriting systems. Technical Report CS-R9073, Centrum voor Wiskunde en Informatica, Amsterdam, 1990.

[KM69] R. Karp and R. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3:147–195, 1969.

[KM02a] A. Kucera and R. Mayr. On the complexity of semantic equivalences for pushdown automata and BPA. In *MFCS: Symposium on Mathematical Foundations of Computer Science*, 2002.

[KM02b] A. Kucera and R. Mayr. Weak bisimilarity between finite-state systems and BPA or normed BPP is decidable in polynomial time. *Theoretical Computer Science*, 270, 2002.

[KM02c] A. Kučera and R. Mayr. Why is simulation harder than bisimulation? *Lecture Notes in Computer Science*, 2421:594, 2002.

[Knu64] D. E. Knuth. Backus Normal Form vs. Backus Naur Form. *Communications of the ACM*, 7(12):735–736, 1964.

[Kön36] D. König. *Theorie der endlichen und unendlichen Graphen.* Akademische Verlagsgesellschaft, Leipzig, 1936.

[KRS04] M. Kretínský, V. Rehák, and J. Strejcek. On extensions of process rewrite systems: Rewrite systems with weak finite-state unit. *Electronic Notes Theoretical Computer Science*, 98:75–88, 2004.

[KRS06] M. Kretínský, V. Rehák, and J. Strejcek. Refining the undecidability border of weak bisimilarity. *Electronic Notes in Theoretical Computer Science*, 149(1):17–36, 2006.

[KS90] P. C. Kanellakis and S. A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Inf. Comput*, 86(1):43–68, May 1990.

[Lin68] A. Lindenmayer. Mathematical models for cellular interaction in development. *J. Theoret. Biology*, 18:280–315, 1968.

[May97] R. Mayr. Combining Petri Nets and PA-Processes. *Lecture Notes in Computer Science*, 1281:547–561, 1997.

[May00a] R. Mayr. On the complexity of bisimulation problems for basic parallel processes. In *ICALP: Annual International Colloquium on Automata, Languages and Programming*, 2000.

[May00b] R. Mayr. Process rewrite systems. *INFCTRL: Information and Computation (formerly Information and Control)*, 156, 2000.

[May02] R. Mayr. Weak bisimilarity and regularity of BPA is EXPTIME-hard. Technical report, Institut für Informatik, Universität Freiburg, December 2002.

[May03a] R. Mayr. Undecidability of weak bisimulation equivalence for 1-counter processes. In *ICALP: Annual International Colloquium on Automata, Languages and Programming*, 2003.

[May03b] R. Mayr. Undecidable problems in unreliable computations. *Theoretical Computer Science*, 297, 2003.

[Mil80] R. Milner. *A Calculus of Communicating Systems.* Springer-Verlag, Berlin, 1 edition, 1980.

[Mil81a] R. Milner. A modal characterisation of observable machine behaviour. In E. Astesiano and C. Bohm, editors, *Trees and Algebra in Programming; 6th CAAP*, volume 112 of *LNCS*, pages 25–34. SpringerVerlag, March 1981.

[Mil81b] R. Milner. A modal characterization of observable machine-behaviour. In E. Astesiano and C. Böhm, editors, *Proceedings CAAP '81*, volume 112 of *LNCS*, pages 25–34, Genoa, March 1981. Springer-Verlag.

[Mil88] R. Milner. Operational and algebraic semantics of concurrent processes. *LFCS Report Series*, pages 1–42, February 1988.

[Mil89] R. Milner. *Communication and Concurrency.* International Series in Computer Science. Prentice Hall, 1989. SU Fisher Research 511/24.

[Min67] M. L. Minsky. *Computation: Finite and Infinite Machines.* Prentice-Hall, 1967.

[MJ84] F. L. Morris and C. B. Jones. An early program proof by alan turing. *Annals of the History of Computing*, 6(2):193–143, 1984.

[MMAHRR03] F-J. Martín-Mateos, J-A. Alonso, M-J. Hidalgo, and J-L. Ruiz-Reina. A formal proof of Dickson's Lemma in ACL2. In Moshe Y. Vardi and Andrei Voronkov, editors, *LPAR*, volume 2850 of *Lecture Notes in Computer Science*, pages 49–58. Springer-Verlag, 2003.

[Mol96] F. Moller. Infinite results. In Ugo Montanari and Vladimiro Sassone, editors, *CONCUR '96: Concurrency Theory, 7th International Conference*, volume 1119 of *Lecture Notes in Computer Science*, pages 195–216, Pisa, Italy, 26–29 August 1996. Springer-Verlag.

[Mol00] F. Moller. A taxonomy of infinite state processes. In P. Jančar and Mojmir Kretinsky, editors, *Electronic Notes in Theoretical Computer Science*, volume 18. Elsevier Science Publishers, 2000.

[MS72] A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *FOCS: IEEE Symposium on Foundations of Computer Science (FOCS)*, 1972.

[MS98] F. Mu and J. Srba. Comparing the classes BPA and BPA with deadlocks. Technical report, December 1998.

[MS05] Y. Matiyasevich and G. Sénizergues. Decision problems for semi-thue systems with a few rules. *Theoretical Computer Science*, 330(1):145–169, 2005.

[MSS04] F. Moller, S. A. Smolka, and J. Srba. On the computational complexity of bisimulation, redux. *INFCTRL: Information and Computation (formerly Information and Control)*, 194, 2004.

[OH86] E. R. Olderog and C. A. R. Hoare. Specification-oriented semantics for communicating processes. *Acta Informatica*, 23:9–66, 1986.

[Pap94] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[Par66] R. J. Parikh. On context-free languages. *Journal of the ACM*, 13(4):570–581, October 1966.

[Par81] D. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Theoretical Computer Science: 5th GI-Conference*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183, Karlsruhe, Germany, 1981. Springer-Verlag.

[Pet62] C. A. Petri. *Kommunikation mit Automaten.* PhD thesis, Schriften des IIM Nr. 2, Bonn, 1962.

[Plo81] G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Department of Computer Science, Aarhus University, Denmark, 1981.

[Plo04] G. D. Plotkin. The origins of structural operational semantics. *Journal of Logic and Algebraic Programming (JLAP)*, 60:3–15, 2004.

[Pos43] E. Post. *Formal Reductions of the General Combinatorial Problem.* American Journal of Mathematics 65. 1943.

[Pre29] M. Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. *Sprawozdanie z I Kongresu Matematikow Krajow Slowcanskich Warszawa*, pages 92–101, 1929.

[Pru90] P. Prusinkiewicz. *The Algorithmic Beauty of Plants (The Virtual Laboratory).* Springer-Verlag, 1990.

[PT87] R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SICOMP: SIAM Journal on Computing*, 16, 1987.

[R̈95] C. Röckl. Proof tableaux for basic parallel processes. Technical report, Fakultät für Informatik, Technische Universität München, 1995.

[Red65] L. Redei. *The Theory of Finitely Generated Commutative Semigroups.* Oxford University Press, 1965.

[Rei85] W. Reisig. *Petri Nets: an introduction.* Springer-Verlag, 1985.

[Rei04] K. Reinhardt. Reachability in Petri Nets with inhibitor arcs. Wilhelm-Schickhard Institut für Informatik, Universitat Tubingen, April 2004.

[Rog67] H. Rogers, Jr. *Theory of Recursive Functions and Effective Computability.* McGraw-Hill, 1967.

[SČ02] J. Střībrná and I. Černá. Modifications of expansion trees for weak bisimulation in BPA. *Electronic Notes Theoretical Computer Science*, 68(6), 2002.

[Sch01] P. Schnoebelen. Bisimulation and other undecidable equivalences for lossy channel systems. In N. Kobayashi and B. C. Pierce, editors, *Proceedings of the 4th International Workshop on Theoretical Aspects of Computer Software (TACS'01)*, volume 2215 of *Lecture Notes in Computer Science*, pages 385–399, Sendai, Japan, October 2001. Springer-Verlag.

[Seg71] K. Segerberg. *An essay in classical modal logic.* Uppsala University, Filosofiska Studier, 13, 1971.

[Sén97] G. Sénizergues. The equivalence problem for deterministic pushdown automata is decidable. In *ICALP: Annual International Colloquium on Automata, Languages and Programming*, 1997.

[Sén98] G. Sénizergues. Decidability of bisimulation equivalence for equational graphs of finite out-degree. In IEEE, editor, *39th Annual Symposium on Foundations of Computer Science: proceedings: November 8–11, 1998, Palo Alto, California*, pages 120–129, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1998. IEEE Computer Society Press.

[Srb02a] J. Srba. Note on the tableau technique for commutative transition systems. In *FOSSACS: International Conference on Foundations of Software Science and Computation Structures*, volume 2303, pages 351–371. Lecture Notes in Computer Science, 2002.

[Srb02b] J. Srba. Roadmap of infinite results. *Bulletin of the European Association for Theoretical Computer Science*, 78:163–, October 2002. Columns: Concurrency. Up-to-date version at `www.brics.dk/~srba/roadmap`.

[Srb02c] J. Srba. Strong bisimilarity and regularity of basic parallel processes is PSPACE-hard. *Lecture Notes in Computer Science*, 2285:535, 2002.

[Srb03] J. Srba. Completeness results for undecidable bisimilarity problems. 2003. Submitted to *INFINITY'03*.

[Sti98a] C. Stirling. Decidability of bisimulation equivalence for normed pushdown processes. *Theoretical Computer Science*, 195, 1998.

[Sti98b] C. Stirling. The joys of bisimulation. *Lecture Notes in Computer Science*, 1450:351–371, 1998.

[Sti01a] C. Stirling. Bisimulation and language equivalence, May 18 2001.

[Sti01b] C. Stirling. Decidability of DPDA equivalence. *Theoretical Computer Science*, 255, 2001.

[Sti01c] C. Stirling. Decidability of weak bisimilarity for a subset of basic parallel processes. *Lecture Notes in Computer Science*, 2030:379–393, 2001.

[Sti01d] C. Stirling. *Modal and Temporal Properties of Processes.* Texts in Computer Science. Springer-Verlag, 2001.

[Stř99] J. Stříbrná. *Decidability and complexity of equivalences for simple process algebras.* PhD thesis, The University of Edinburgh, 1999. Also available as The University of Edinburgh, LFCS report ECS-LFCS-99-408.

[Sus03] M. Sustik. Proof of Dickson's Lemma using the ACL2 theorem prover via an explicit ordinal mapping. In *Workshop on the ACL2 Theorem Prover and Its Applications*, July 2003.

[Tar51] A. Tarski. *A Decision Method for Elementary Algebra and Geometry.* Univ. of California Press, 2nd edition, 1951.

[Tau89] D. Taubner. *Finite Representations of CCS and TCSP Programs by Automata and Petri Nets*, volume 369 of *Lecture Notes in Computer Science.* Springer-Verlag, 1989.

[Thu14] A. Thue. Probleme über Veränderungen von Zeichenreihen nach gegebenen Regeln. *Skr. Vid. Kristianaia I. Mat. Naturv. Klasse*, 10/34, 1914.

[Var01] M. Vardi. Branching vs. linear time: Final showdown. In *TACAS: International Workshop on Tools and Algorithms for the Construction and Analysis of Systems, LNCS*, 2001.

[vB76] J. F. van Benthem. *Modal correspondence theory.* PhD thesis, Mathematical Institute, University of Amsterdam, Amsterdam, NL, 1976.

[vG90] R. J. van Glabbeek. The linear time-branching time spectrum (extended abstract). In *CONCUR*, pages 278–297, 1990.

[vG94] R. J. van Glabbeek. What is branching time semantics and why to use it? In M. Nielsen, editor, *The Concurrency Column*, pages 190–198. *Bulletin of the EATCS* 53, 1994. Also available as Report STAN-CS-93-1486, Stanford University, 1993, at `http://-theory.stanford.edu/branching/`, and in G. Paun, G. Rozenberg & A. Salomaa, editors: *Current Trends in Theoretical Computer Science; Entering the 21st Century*, World Scientific, 2001.

[vG01] R. J. van Glabbeek. The linear time – branching time spectrum I. The semantics of concrete, sequential processes. In J. A. Bergstra,

A. Ponse, and S. A. Smolka, editors, *Handbook of Process Algebra*, pages 3–99. North-Holland, 2001.

[vGW96] R. J. van Glabbeek and W. Weijland. Branching time and abstraction in bisimulation semantics. *JACM: Journal of the ACM*, 43, 1996.

# Glossary

| | |
|---|---|
| $\aleph$ | ranges over the cardinal numbers. |
| $\alpha, \beta, \gamma, \delta$ | are some BPP or BPA processes. |
| $\sim$ | is the largest strong bisimulation relation: our chosen notion of behavioural equivalence between processes (page 45). |
| $\approx$ | is the largest weak bisimulation relation. *Weak* bisimilarity ignores silent or internal action (page 52). |
| $\approx_\kappa$ | is the $\kappa$th weak approximant, it approximates $\approx$ in the sense that it considers $\kappa$ steps where $\approx$ takes account of *all* steps (page 52). |
| $\approx^!_\kappa$ | means *exactly* $\kappa$ equivalent: $u \approx_\kappa v$ and $u \not\approx_{\kappa+1} v$. |
| BPA | Basic Process Algebra, the non-commutative context-free processes; equivalent to $\epsilon$-free, single control-state PDA (pages 29, 59). |
| BPP | The Basic Parallel Processes, the commutative context-free processes; equivalent to the communication-free Petri Nets (pages 15, 32, 60). |
| $\mathcal{C}$ | is the class of cardinal numbers (page 22). |
| $d_a(u)$ | is the minimum number of transitions from $u$ to a process $v$ which cannot perform an $a$ action, $v\overset{a}{\not\to}$. Note that $d_a = dd(a, \emptyset)$. |
| $d^\omega_{a_\tau}(u)$ | is the minimum number of weak transitions $(\overset{a}{\Rightarrow})$ from $u$ to a process $v$ which cannot perform an arbitrary number of $a$ transitions, $\exists n.v\overset{a^n}{\not\Rightarrow}$ (page 89). |
| $dd(a, D)$ | is a distance-to-disablings function. $dd(a, D)(u)$ is the minimum distance from $u$ to a process $v$ for which $v\overset{a}{\not\to}_D$, where $D$ is itself composed of distances-to-disablings-$\delta$ pairs (page 80). |

| | |
|---|---|
| $dd^{\kappa}_{\tau}(a, D)$ | is an extended weak distance-to-disablings function. $dd^{\kappa}_{\tau}(a, D)(u)$ is the minimum weak distance (plus one, if $u \neq v$) to a process $v$ which cannot perform a sequence $\kappa$ long of $\overset{a}{\Rightarrow}_D$ transitions, $v \overset{a^{\kappa}}{\not\Rightarrow}_D$ (page 90). |
| $\mathrm{DD}^{\aleph}$ | is the set of distances-to-disablings whose constituent sets are bound strictly above by $\aleph$ (page 80). |
| DD | abbreviates $\mathrm{DD}^{\aleph_0}$ (the finite distances-to-disablings functions). |
| $\mathrm{DD}^{\mathcal{C}}$ | is the full class of distances-to-disablings functions, with no restriction as to the size of sets used in their construction (page 80). |
| $\mathrm{DD}_{\tau}$ | is the set of finite weak distances-to-disablings functions (page 86). |
| $\mathrm{DD}'_{\tau}$ | is $\mathrm{DD}_{\tau}$ augmented with $d^{\omega}_{a_{\tau}}$ (page 90). |
| $\mathrm{DD}^{\omega}_{\tau}$ | are the finite extended weak distances-to-disablings functions $dd^{\kappa}_{\tau}(a, D)$ for which the $\kappa$ part is either 1 or $\omega$ (page 90). The conjecture of this thesis is that these functions express weak bisimilarity on BPP processes (page 102). |
| $\equiv_{\mathrm{DD}^{\mathcal{C}}}$ | means $u \equiv_{\mathrm{DD}^{\mathcal{C}}} v$ iff $d(u) = d(v)$ for all $d \in \mathrm{DD}^{\mathcal{C}}$ (page 80). |
| $\mu, \kappa$ | range over the ordinal numbers. |
| $\mathbb{N}_{-1,\infty}$ | $= \mathbb{N} \cup \{-1, \infty\}$ |
| $\mathcal{O}$ | is the class of ordinal numbers (page 19). |
| $u \xrightarrow{a} v$ | means a process $u$ performs an $a$ action and becomes a process $v$; equivalently, in a labelled transition system there is an $a$-labelled arrow from a node $u$ to a node $v$ (page 9). |
| $u \xrightarrow{a}_{d,\delta} v$ | means $u \xrightarrow{a} v$ and $d(u) + \delta = d(v)$ (page 79). |
| $u \xrightarrow{a}_{D} v$ | Where $D$ is a set of $(d, \delta)$ pairs, means $u \xrightarrow{a}_{d,\delta} v$ for all $(d, \delta) \in D$; $u \overset{a}{\not\to}_D v$ if either $u \overset{a}{\not\to} v$ or $\exists (d, \delta) \in D$ s.t. $d(u) + \delta \neq d(v)$ (page 79). |
| $u \overset{a}{\Rightarrow} v$ | means a process $u$ performs some number of silent transitions, then a strong $a$ action, then some number of silent transitions, to become the process $v$; $u \xrightarrow{\tau^*} \xrightarrow{a} \xrightarrow{\tau^*} v$ (page 11). |

# Index

$DOM(\vec{x})$, **35**
$G(u,v)$, **46**
$LEX(\vec{x})$, **36**
$S$, **40**
DD, **80**
  finite approximants, **84**
$\mathrm{DD}'_\tau$, **89**
  BPP, **98**
$\mathrm{DD}_\tau$, **87**
  BPP
    difficulties, **92**
    insufficiency, **96**
  extending, **88**
  finite approximability, **95**
$\mathrm{DD}^{\mathcal{O}}_\tau$
  collapse, **98**
$\mathrm{DD}^{\mathcal{C},\mathcal{O}}_\tau$
  collapse, **91**
$\mathrm{Reach}_\tau$, **72**
$\Sigma$, **40**
$\equiv_{\tau LT}$, **52**
$\equiv_{\tau L}$, **52**
$\longrightarrow$, **40**

approximants, **47**
  finite approximants for infinite alphabets, **49**
  hierarchy, **49**
    $\omega^\omega$ on BPP, **76**
    collapse, **51**

bisimilarity, **44**
  between BPA and BPP, **62**
  context-free processes, **65**
  game, **46**
  other notions, **54**
BPA, **59**
  fcBPA, **70**
BPP, **60**
  approximant collapse, **75**
  fcBPP, **70**
  finite weak approximants are decidable, **73**
  purely generated, **71**
  semidecidability of weak bisimilarity, **71**
  silently lossy, **65**
  single-action, non-norm-zero, **96**
  totally normed, **71**

cardinals, **22**
  regular, **23**
Caucal bases, **65**
Chapter
  1, Introduction, **7**
  2, Preliminaries, **19**
  3, Constructive Dickson's Lemma, **34**
  4, General processes, **40**
  5, Processes from context-free grammars, **58**
  6, Distance functions, **78**
  7, Distance functions on BPP, **92**
  8, A research programme, **100**
congruences, **24**

Conjectures
  $\mathrm{DD}^{\omega}_{\tau}$, **99**
  $\equiv_{\mathrm{DD}_\tau}$
    undecidable on BPP, **97**
  $\approx \subsetneq \equiv_{\mathrm{DD}'_\tau}$, **98**
  BPA approximant collapse, **76**
  BPP approximant collapse, **76**
Corollaries
  $\mathrm{DD}'_\tau$
    BPA, **90**
  $\approx_n$ as $\sim$ on finite processes, **74**
  Caucal bases, **66**
  commutative regular language star height, **29**
  congruences on $(\mathbb{N}^k, +)$, **27**

Definitions
  $DD_{a,D}$, **95**
  $D^{\infty}$, **94**
  $D^n$, **94**
  $D_a$, **94**
  $G^{\tau}_{\kappa}$, **52**
  DD
    height, **80**
  $\mathrm{DD}'_\tau$, **89, 90**
  $\mathrm{DD}^{\mathcal{C}}$, **80**
    countable, **80**
    finite, **80**
  $\mathrm{DD}^{\mathcal{C}}_{\tau}$, **86**
  $\Delta'(\alpha, n, \beta)$, **93**
  $\Delta(\alpha, n, \beta)$, **93**
  $\Delta_\phi$, **94**
  $\overset{R}{\equiv}$, **25**
  dist, **78**
  $\mathrm{dist}_\tau$, **86**
  $\equiv_{\mathrm{DD}^{\mathcal{C}}}$, **80**
  $\equiv_{\tau LT}$, **52**
  $\equiv_{\tau L}$, **52**
  $\phi_{a,D}$, **94**
  $\sim \kappa$, **47**
  $\xrightarrow{a}_F$, **79**
  $\xrightarrow{w}$, **41**
  $\overset{a^\kappa}{\Rightarrow}_F$, **89**
  $d_a$, **79**
  $d^{\omega}_{a}$, **89**
  $d^{\omega}_{a_\tau}$, **89**
  $k$-tuple, **24**
  $r_n$, **38**
  *ugly* weak distance, **86**
  approximants, **47**
  bisimulation, **45**
    game, **46**
  BPP
    very simple, **68**
  branching bisimulation, **55**
  Caucal bases, **65**
  domination, **34**
  equivalence relation
    congruence, **24**
  equivalence relation, **24**
  expansion, **66**
  failures and readiness, **48**
  failures equivalence, **44**
  graph equivalence, **43**
  Hirshfeld tree, **67**
  labelled transition systems, **40**
  linear set, **25**
  monoids, **23**
  norm, **41**
  norm reducing, **42**
  optimal move trees, **50**
  optimal moves, **50**
  ordinals
    von Neumann's, **20**
  Parikh mapping, **24**
  Petri Nets

communication-free, **32**
Presburger arithmetic
computable, **26**
sets, **26**
processes
$k$-tuple, **42**
finite, **46**
general, **40**
normed, **41**
sub-$\aleph$-branching, **42**
void, **41**
well-founded, **41**
pumping path, **28**
pure, **28**
semilinear sets, **26**
sub-$\aleph$-branching, **23**
T operator, **53**
trace equivalence, **43**
trees, **21**
height, **21**
largest domination-free tree, **35**
well-founded, **21**
vector, **24**
weak bisimilarity
weak bisimulation relation, **52**

domination, **34**
reification, **38**

equivalence relations, **24**
functional, **43**
Example
$\mathrm{DD}_{\mathcal{T}}$
trace, failures equivalence, **95**
Examples
$(\Sigma^*, \cdot)$, **24**
$\mathrm{DD}_{\mathcal{T}}$
finite alphabets, **87**
$\mathrm{DD}_{\mathcal{T}}^{\mathcal{O}}$, **90**
$\mathrm{DD}_{\mathcal{T}}^{\mathcal{C},\mathcal{O}}$ on weakly-finitely branching processes, **91**
$\equiv_S$ as a finite automaton, **27**
$\sim_\omega$, **47**
$d_{a_\mathcal{T}}$
properties, **93**
$d^\omega_{a_\mathcal{T}}$, **90**
$u \xrightarrow{a}_{d,\delta}$, **79**
$u \overset{\epsilon}{\Rightarrow} v \overset{\epsilon}{\Rightarrow} u$, **52**
$\sim_\omega \subsetneq \equiv_L$, **48**
$\sim \subseteq \equiv_{LT}$, **46**
approximants
collapse on finitely-branching processes, **50**
collapse on well-founded processes, **51**
bisimilarity
bisimilar and non-bisimilar processes, **45**
BPP, **60**
lossy, **65**
failures and readiness, **48**
finitely generated sum congruence, **25**
König's Lemma, **23**
languages, **24**
maximal norms, **61**
monoids
finitely generated, and non-finitely generated, **25**
ordinals
$\omega$ as "arbitrarily far", **19**
proper class, **22**
Petri Nets, **30**
semilinearity of $\equiv_S$, **26**
sum equivalence, **25**
trees
addition, **22**

sub-$\aleph$-branching, **23**
weak bisimilarity
single-step weak bisimulations, **53**
extensions of context-free processes, **64**

finite automata, **27**

generalised continuum hypothesis, **49**
grammars as processes, **11**, **59**

Hirshfeld trees, **66**

Jančar's distances to disablings, **68**

labelled transition systems, **40**
deterministic, **42**
unexplorable finitely-branching, **48**
languages
unary, **63**
Lemmas
$DD_{a,D}$, **95**
$D_a$, **94**
$S_n$, **38**
DD
$\sim_n$, **84**
$\mathrm{DD}'_\tau$
BPP, **98**
$\mathrm{DD}^{\aleph}$
approximant collapse, **84**
$\mathrm{DD}^{\mathcal{C}}$, **80**
$\mathrm{DD}_\tau$
normed BPP, **97**
$\mathrm{DD}'_\tau$, **89**
$\mathrm{DD}^{\mathcal{C}}_\tau$, **86**
$\mathrm{DD}^{\mathcal{C}}_{wn}$, **85**
$\Delta$, **93**
NORM, **69**
$\psi$, **72**
$d_a^\infty$, **98**
$r_n$, **38**
$G_\kappa = \sim_\kappa$, **47**
approximants
collapse on sets of processes, **51**
finite approximants with finite alphabets, **49**
hierarchy, **49**
sub-$\aleph$-branching collapse, **51**
bisimilarity
deterministic processes, **45**
finite processes, **46**
BPP
$\approx$ and $\approx_\kappa$ are congruences, **70**
$\approx$ is semidecidable, **72**
domination, **76**
finite approximant branching, **74**
pumping lemma, **61**
single-action, non-zero-norm, **96**
transition sequences, **32**
cardinals
no largest cardinal, **23**
Caucal bases, **66**
Dickson's Lemma, **35**
optimal move trees, **50**
ordinals
well-founded trees, **22**
PDA
control-state hierarchy, **63**
Petri Nets
transition sequences, **32**
Reach, **32**
semideciding $\not\sim$ on finitely branching processes, **48**
semilinear sets
membership, 26
T operator, **54**
trees
lexicographically-ordered, **36**
Ugly lemma, **86**

LTS, **40**

machines, **29**
monoids, **23**
    free commutative, **24**

norm reducing transition, **42**

Open problems
    $DD_\tau$, **101**
    $DD'_\tau$, **102**
    $DD^{\omega\times2}_\tau$, **102**
    $DD^\omega_\tau$, **102**
optimal move trees, **50**
ordinals, **19**
    arithmetic, **20**
    Cantor normal form, **20**
    limit, **19**
    successor, **19**
organisation, **17**

Parikh mapping, **24**
Petri Nets, **30**
    communication-free, **32**
Presburger arithmetic, **26**
processes
    equivalence relations, 42
    finite, **46**
pushdown automata, **29**

reification, **37**

semideciding $\not\approx$, **48**
semilinear sets, **25**
silent actions, **51**

tableau proofs, **67**
Theorems
    $G^\tau$, **52**
    $DD_\tau$
        computable on BPP, 95
    bisimilarity
        game, **46**
    BPP
        approximant collapse, **77**
        finitely approximable, **73**
    Cantor normal form, **20**
    comparing BPA and BPP, **62**
    congruences on finitely generated monoids, **26**
    Eilenberg and Schützenberger, **26**
    Ginsburg and Spanier, **26**
    Language equivalence is undecidable on context-free grammars, **58**
    Presburger arithmetic
        equals the semilinear sets, **26**
        is decidable, **26**
    Redei, **25**
    semilinear sets
        finite automata equivalence, **28**
    trees
        non-dominating vector-labelled, **39**
        sub regular cardinal-branching, **23**
total order, **36**
trees, **21**
    domination-free, **35**
    well-founded, **21**

vector space, **24**
void, **41**
    $\text{void}_\tau$, **54**

weak bisimilarity, **52**
    context-free processes, **70**
weak distance, **85**
    refining, 86