



Basic Research in Computer Science

BRICS RS-95-25

S. B. Lassen: Basic Action Theory

Basic Action Theory

Søren B. Lassen

BRICS Report Series

RS-95-25

ISSN 0909-0878

May 1995

**Copyright © 1995, BRICS, Department of Computer Science
University of Aarhus. All rights reserved.**

**Reproduction of all or part of this work
is permitted for educational or research use
on condition that this copyright notice is
included in any copy.**

**See back inner page for a list of recent publications in the BRICS
Report Series. Copies may be obtained by contacting:**

**BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK - 8000 Aarhus C
Denmark
Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk**

**BRICS publications are in general accessible through WWW and
anonymous FTP:**

**<http://www.brics.dk/>
[ftp ftp.brics.dk \(cd pub/BRICS\)](ftp://ftp.brics.dk/cd/pub/BRICS)**

Basic Action Theory ^{*}

S. B. Lassen

BRICS[†]

Department of Computer Science

University of Aarhus

email: thales@brics.aau.dk

URL: <http://www.daimi.aau.dk/~thales>

Abstract

Action semantics is a semantic description framework with very good pragmatic properties but until now a rather weak theory for reasoning about programs. A strong action theory would have a great practical potential, as it would facilitate reasoning about the large class of programming languages that can be described in action semantics.

This report develops the foundations for a richer action theory, by bringing together concepts and techniques from process theory and from work on operational reasoning about functional programs. Semantic preorders and equivalences in the action semantics setting are studied and useful operational techniques for establishing contextual equivalences are presented. These techniques are applied to establish equational and inequational action laws and an induction rule.

^{*}A preliminary version was presented at the 6th Nordic Workshop on Programming Theory, Aarhus, October 17–19, 1994, and appeared in the workshop proceedings.

[†]Basic Research in Computer Science, Centre of the Danish National Research Foundation.

Contents

1	Introduction	1
2	Action Semantics	3
2.1	Action semantic descriptions	3
2.2	Action notation	4
2.3	Basic actions	4
2.4	Action laws	6
3	Reduction semantics	7
3.1	Decomposition	7
3.2	Evaluation	8
3.3	Properties of evaluation	9
4	Contextual testing	11
4.1	Definedness	11
4.2	Contextual testing equivalence	12
4.3	Basic action theory	13
5	Finite testing	16
5.1	Open extension	16
5.2	Precongruence	17
5.3	Soundness of open extension	18
6	Alternative characterisation	20
6.1	Experimental order proof technique	21
6.2	Basic context lemma	24
6.3	Stronger characterisations	27
7	Simulation	28
7.1	may and must simulation	28
7.2	Simulation proof techniques	30
7.3	Simulation versus contextual testing	32
7.4	Identity of all characterisations	34
8	(In)equational action theory	35
8.1	Equational theory	35
8.2	Inequational theory	37
9	Stuck actions	39
10	Generalisations to other facets	40
11	Conclusion	42

References	44
Index of symbols	47

1 Introduction

In this report we develop a richer theory for reasoning about programs in *action semantics* (AS). Because AS is a general semantic description framework, our work has a wide practical scope. A strong action theory would offer techniques for reasoning about programs in any programming language that can be described in AS.

AS is a denotational-style semantics. An *action semantic description* (ASD) maps program terms to the semantic entities they denote. The semantic entities are called *actions*. ASDs are compositional, therefore language terms are contextually equivalent if their denotations, i.e. actions, are equivalent by a suitable notion of action equivalence.

The undertaking of this report is to examine notions of semantic equivalence of actions and techniques for reasoning about these action equivalences.

Existing action theory

Action theory has hitherto mainly consisted of a series of action laws that codify algebraic properties of actions. Mosses [22, App.C] defines a testing equivalence (a sort of **may** equivalence [13]), based on a structural operational semantics for actions, and develops a bisimulation proof technique. By means of bisimulation, (almost) all action laws can be shown to respect testing equivalence.

This report reworks and strengthens the foundations of Mosses' approach and rectifies some deficiencies therein. Moreover, we obtain a richer theory by working with a range of preorders and equivalences.

Reasoning with actions

Actions have functional, imperative, as well as concurrent features. There is thus a great range of places to look for inspiration for developing action theory. This report treats a subset of actions focussing on nondeterminism and parallelism but all the theory is set up with a view to future extension to the full set of actions.

Process theory Testing [4] and bisimulation [31] are notions widely used in concurrency. Our definitions of preorders and equivalences for actions are based on testing and some of our proof techniques use (bi)simulation.

Bisimulation is normally the strongest equivalence considered in semantics and is, generally, included in all other equivalences. We shall consider weak bisimulation that abstracts away from unobservable, internal computation steps. Standard weak bisimulation doesn't guarantee test equivalence, however, because it is insensitive to internal divergence. Therefore we use refined, divergence-sensitive (bi)simulations [35, 34, 28].

Operational reasoning about functional programs In recent years, bisimulation has been subject to much study in functional settings, mostly for pure, deterministic languages but also in conjunction with input/output, state, nondeterminism, or parallelism. (Gordon [10] explains and summarises research in this area.) This research has, however, paid scant attention to testing equivalence in the presence of nondeterminism. In this report we wish to develop a testing theory for actions and therefore take a different approach as starting point, namely Ian Mason and Carolyn Talcott’s operational framework for reasoning about impure functional languages which lends itself better to testing. They have developed powerful operational techniques for deciding program equivalences that involve memory [19, 14] and have also applied their techniques to a concurrent **Actor** language [1].

A drawback of these functional approaches is that they deal with particular, idealised functional languages. Bisimulation techniques seem to be applicable to most operationally defined languages but a considerable amount of work is required to set up the framework. Mason and Talcott have treated a number of programming concepts but it is not clear how to apply their techniques to other programming languages in general—the operational semantics must be of a certain form (does this constrain the class of languages that can be considered?) and, for every language, comprehensive efforts are required to develop the basic theory.

In contrast, if we can develop a strong action theory, it will apply readily to the large class of programming languages that admit an ASD.

Why actions? It may appear paradoxical why we should not be content with a strong theory of λ -calculi and functional languages since λ -calculi are used extensively as meta-language for denotational semantics. However, for large programming languages, the encoding of semantics as higher-order functions, dictated by λ -calculi, tends to yield inaccessible semantic descriptions that are hard to maintain and extend. Actions have been designed to rectify these deficiencies and experience with action semantics substantiates this claim.

We shall in the following apply the various operational techniques to develop an inequational and equational theory for actions. We have managed to fit actions into Mason and Talcott’s framework and to generalise their techniques to nondeterminism as it appears in actions. On the basis of this, we have moreover developed more elegant simulation proof methods.

In what follows only the results for a small subset of actions are reported. The subset serves to illustrate the major interesting points of the theory. We have extended these results to substantial parts of the language of actions but a full presentation becomes very lengthy. We plan to report on further portions of our work in the near future.

Overview

Section 2 introduces AS as well as the ‘basic’ actions and action laws to be treated in the rest of the report. In order to fit actions into Mason and Talcott’s framework, Section 3 defines a so-called reduction semantics for actions. In Section 4, contextual testing preorders and equivalences are defined and the equivalence classes are determined. Section 5 and Section 6 establish a context lemma and, based on this, develops a technique for proving actions preordered. Section 7 develops simulation proof methods which are used to sketch an inequational and equational action theory in Section 8 in terms of action laws and an induction rule. In Section 9 the role of ‘stuck’ actions is discussed. Section 10 surveys the generalisation of the theory to arbitrary actions. A discussion of future and related work concludes the report.

2 Action Semantics

Action semantics (AS) is a semantic description framework with very good pragmatic properties [22]. It has been used to formalise a wide range of realistic programming languages, as demonstrated by complete descriptions of Pascal [25], ANDF [12], most of Occam2 [3], as well as numerous other functional, procedural, object-oriented, and parallel languages. Moreover, a number of compiler generators have been based on AS [30, 5, 27, 29].

2.1 Action semantic descriptions

An action semantic description (ASD) consists of compositional semantic functions that map abstract syntax to the universal domain of actions. Suppose we have an imperative language with a syntactic category **Statement** in the abstract syntax. We can have a semantic function such as:

- $\text{execute } _ :: \text{Statement} \rightarrow \text{action} .$

Some example semantic equations:

- (1) $\text{execute } \text{“skip”} = \text{complete} .$
- (2) $\text{execute } \llbracket S_1:\text{Statement } \text{“;” } S_2:\text{Statement} \rrbracket =$
 $\text{execute } S_1 \text{ and then execute } S_2 .$
- (3) $\text{execute } \llbracket \text{“loop” } S:\text{Statement} \rrbracket =$
(unfolding
(execute S and then unfold))
trap complete .
- (4) $\text{execute } \text{“exit”} = \text{escape} .$

The actions appearing on the right hand sides of the semantic equations will be explained shortly.

2.2 Action notation

Action notation (AN) is the specification language of AS for expressing actions. It is a rich language which expresses common computational notions in a direct way. AN is structured into various *facets* according to the information being processed:

- The **basic** facet is concerned with *control*, no information is processed. Higher-order, exceptional, nondeterministic, and interleaving control flow are expressible.
- The **functional** facet—processes transient data; features unbounded choice.
- The **declarative** facet—deals with bindings; control, data, and binding flow may be combined in various ways.
- The **imperative** facet—operates on a single-threaded storage.
- The **communicative** facet—connects distributed agents which communicate asynchronously.

The semantic equations above employ only the basic facet of AN. For simplicity we shall focus on the basic actions in this report. The operational machinery we shall present is currently developed for all but the communicative facet.

2.3 Basic actions

AN is a combinator-based notation. Actions are either *primitives* or compounds built using *combinators*. The outcome of performing an action is:

completed, normal termination (subsequent processing continues normally);

escaped, exceptional termination (transfers control to an exception handler);

failed, abnormal termination (aborts current processing); or

divergent, no termination.

The basic primitives are:

complete, **escape**, and **fail**, completes, escapes, and fails, respectively.

check $_$, takes a truth value argument and completes if it is true and fails if it is false.

enact $_$, invokes the abstraction yielded by its argument—an abstraction is an abstracted (or ‘reified’) action.

unfold, marks a point of recursion in the body of the unary combinator **unfolding** (see below).

The basic combinators are:

and then, binary sequential combinator.

and, binary combinator; performs its two subactions in any, arbitrarily interleaved order.

trap, binary combinator; performs its second subaction as an exception handler if the first subaction escapes.

or, binary choice combinator; if one subaction fails, the other is chosen, if neither subaction fails, one is chosen nondeterministically.

unfolding, unary recursion combinator; the primitive **unfold** in the subaction of **unfolding** denotes a recursive unwinding of the subaction. A divergent action is most shortly written **unfolding unfold** . (Let **diverge** abbreviate this action.)

A grammar of basic AN can be specified in unified algebras [24] as follows (the vertical bar denotes sort union):

- **action-primitive** = **complete** | **escape** | **fail** | **check yielder** | **enact yielder** | **unfold** .
- **action** = **action-primitive** | **action and then action** | **action and action** | **action trap action** | **action or action** | **unfolding action** .
- **yielder** = **truth-value** | **abstraction** .

The arguments to the parameterised primitives **check** and **enact** are called yielders. In the basic facet, yielders are just constants: either truth values,

- **truth-value** = **true** | **false** ,

or abstractions, built from actions using the constructor **abstraction of** $_$,

- **abstraction of** $_$:: **action** \rightarrow **abstraction** .

Open and closed actions

An action A is *closed* if it has no ‘free’ occurrences of the primitive **unfold**, i.e. occurrences not enclosed by the unary combinator **unfolding**. Otherwise it is *open*. Let **closed-action** denote the **action** subsort consisting of all closed actions. Open actions are only meaningful as subactions of **unfolding**. Only closed actions can be performed.

The operator $_@_$ substitutes its second argument for all free occurrences of the primitive **unfold** in the first argument, ($A_1@A_2 = “A_1[A_2/\text{unfold}]”$).

- $_@_$:: **action**, **action** \rightarrow **action** (*associative, unit is unfold*) .

- (1) $\text{unfold}@U = U$.
- (2) $(\text{unfolding } A)@U = \text{unfolding } A$.
- (3) $O(A_1 \dots A_n)@U = O(A_1@U \dots A_n@U)$, if $O \neq \text{unfolding}$.¹

(A, A_i, U range over actions and O is any action construct other than **unfolding**.)
 $A_1@A_2$ is closed if A_1 or A_2 is. An action A is closed if and only if $A@U = A$, for all actions U .

Action denotations

To illustrate how actions give semantics to programming language terms, consider two statements from the example language from before:

- (i) **skip** .
- (ii) **loop exit** .

The semantic function `execute` maps them to a primitive and to a compound action, respectively; these actions are the ‘denotations’ of **skip** and **loop exit**.

- (i) **complete** .
- (ii) (**unfolding**
 (**escape and then unfold**))
 trap complete .

The latter escapes inside the body of **unfolding**, is then caught by **trap**, and the whole action completes. We expect (i) and (ii) to be interchangeable and in Section 2.4 below we shall see that the action denotations are indeed equivalent.

Basic actions are not very expressive in themselves. Basic actions can just diverge or terminate in three different ways. In particular, they cannot process data in any way. Only when fused with data processing facets does the expressive power of basic actions become interesting. Nevertheless, the basic facet suffices to illustrate the main concepts and techniques which we shall present, and which apply equally well to the other facets.

2.4 Action laws

There are a series of ‘action laws’ that characterise actions. The following are a sample of basic action laws:

- (1) **check true = complete ; check false = fail** .

¹Normally, see [22], @ doesn’t substitute into bodies of abstractions. Our definition can be made to coincide with the usual definition by restricting abstraction to closed actions,

- $\text{abstraction} = \text{abstraction of closed-action}$.

- (2) unfolding $A = A @$ unfolding A .
- (3) $(A_1$ and then $A_2)$ and then $A_3 = A_1$ and then $(A_2$ and then $A_3)$.
- (4) complete and then $A = A$ and then complete $= A$.
- (5) escape and then $A =$ escape ; fail and then $A =$ fail .
- (6) $(A_1$ trap $A_2)$ trap $A_3 = A_1$ trap $(A_2$ trap $A_3)$.
- (7) escape trap $A = A$ trap escape $= A$.
- (8) complete trap $A =$ complete ; fail trap $A =$ fail .
- (9) $(A_1$ or $A_2)$ or $A_3 = A_1$ or $(A_2$ or $A_3)$.
- (10) fail or $A = A$ or fail $= A$.
- (11) A_1 or $A_2 = A_2$ or A_1 .
- (12) A or $A = A$.

Law (1) characterises check. Law (2) states that `unfolding` is equal to its own unwinding. Laws (3)–(5) say that `and then` is associative, has `complete` as unit, and is left-absorbed by `escape` and `fail`. Laws (6)–(8) are corresponding laws for `trap`. The `or` combinator is furthermore commutative (11) and idempotent (12).

As an example application of these laws, the example actions (i)–(ii) above can be shown equal using laws (5), (2), (7).

A catalogue of action laws for all facets of AN may be found in [22, App.B]. In abstract semantic algebras [21], a precursor of AS, such laws served as an algebraic semantics of actions. However, establishing laws sufficient to define full action notation in this way doesn't seem feasible.

3 Reduction semantics

Mason and Talcott operate on the basis of a ‘reduction semantics’ for their language. A reduction semantics is a small-step operational semantics that represents control and state syntactically.

Our first step towards fitting actions into Mason and Talcott’s framework is to define a reduction semantics for basic AN. (A preliminary sketch was presented in [16].) The essential novelty compared to [22, App.C] is that control is represented by means of evaluation contexts.

3.1 Decomposition

An *evaluation context* [8] (or *reduction context* [19]) is an action with a hole at a legal point of execution in the action. Either just a hole `[]`, a hole to the left of the sequential combinators (*seq* is either of `and then`, `trap`), or a hole on

either side of the symmetric combinators (*sym* is either of **and** , **or**)—as a BNF grammar:

$$E ::= [] \mid E \text{ seq } A \mid E \text{ sym } A \mid A \text{ sym } E ,$$

where A ranges over actions and E over evaluation contexts.

A *redex* R is any action that can make a transition directly (see the evaluation rules below). $E[R]$ denotes the action obtained by filling R into the hole in E .

In an action $E[R]$, the evaluation context E corresponds to the point of execution in the program, the program pointer, or current continuation. The redex R is the current instruction.

When an action A can be written as an evaluation context E filled with a redex R , viz. $A = E[R]$, we say that A *decomposes* into E, R .

Remark 3.1 Decomposition is not unique. For example, there are two possible decompositions of $A = \text{check true or (check false and then escape)}$,

namely $A = E_1[\text{check true}] = E_2[\text{check false}]$,

where $E_1 = []$ or (check false and then escape),

$E_2 = \text{check true or } ([] \text{ and then escape})$. □

3.2 Evaluation

The evaluation rules decompose an action and replace the redex with a reduced (or unwound) action. Let A_1, A_2 :closed-action, A :action, and Y :yielder,

- (1) $E[\text{check } Y] \mapsto \begin{cases} E[\text{complete}] & \text{if } Y = \text{true} \\ E[\text{fail}] & \text{if } Y = \text{false} . \end{cases}$
- (2) $E[\text{enact } Y] \mapsto E[A_1] \text{ if } Y = \text{abstraction of } A_1 .$
- (3) $E[\text{unfolding } A] \mapsto E[A@\text{unfolding } A]$.
- (4) $E[A_1 \text{ and then } A_2] \mapsto \begin{cases} E[A_1] & \text{if } A_1 : \text{escape} \mid \text{fail} \\ E[A_2] & \text{if } A_1 = \text{complete} . \end{cases}$
- (5) $E[A_1 \text{ and } A_2] \mapsto \begin{cases} E[A_1] & \text{if } A_1 : \text{escape} \mid \text{fail} \text{ or } A_2 = \text{complete} \\ E[A_2] & \text{if } A_1 = \text{complete} \text{ or } A_2 : \text{escape} \mid \text{fail} . \end{cases}$
- (6) $E[A_1 \text{ trap } A_2] \mapsto \begin{cases} E[A_1] & \text{if } A_1 : \text{complete} \mid \text{fail} \\ E[A_2] & \text{if } A_1 = \text{escape} . \end{cases}$
- (7) $E[A_1 \text{ or } A_2] \mapsto \begin{cases} E[A_1] & \text{if } A_1 : \text{complete} \mid \text{escape} \text{ or } A_2 = \text{fail} \\ E[A_2] & \text{if } A_1 = \text{fail} \text{ or } A_2 : \text{complete} \mid \text{escape} . \end{cases}$

Terminology

Let **success** = complete | escape , **terminated** = success | fail .

An action is *final* if there are no transitions out of it. A *computation* from (or ‘performed by’) some action A is a sequence of transitions, $A \mapsto A_1 \mapsto A_2 \mapsto \dots$.

Computations are *successful* and *terminated* if they end in final actions in **success** and **terminated**, respectively.

A non-terminated, final action is *stuck*, so is a computation ending in a stuck action. `enact false` is an example of a stuck action. In the following we shall assume that all actions are *well-behaved*: A closed action is well-behaved if it never gets stuck and all its subterms are well-behaved. (Section 9 discusses how stuck actions fit into our theory.)

An action A *must converge*, written $A \Downarrow$, if all computations from A are finite.

Definition 3.2 \Downarrow is the least predicate on closed actions satisfying

$$A \Downarrow \quad \text{iff, for all } B, \quad A \mapsto B \text{ implies } B \Downarrow .$$

\Uparrow denotes the negation of \Downarrow , $A \Uparrow \stackrel{\text{def}}{=} \neg(A \Downarrow)$.

\Downarrow can be expressed inductively:

$$\Downarrow = \bigcup_{\mu < \omega} \Downarrow^\mu \quad \text{where} \quad \Downarrow^\mu = \{ A \mid (\forall B \mid A \mapsto B) B \in \bigcup_{\nu < \mu} \Downarrow^\nu \} .$$

Basic action notation is finitely branching and by König's lemma we only need to do induction over the finite ordinals (all $\mu < \omega$). If we add the functional facet which introduces countably branching nondeterminism, induction over all recursive ordinals is required [2].

3.3 Properties of evaluation

We need to develop some machinery for reasoning about evaluation. Our observations and results about evaluation rules and transitions are similar to Plotkin's activity lemma [33] and Mason and Talcott's '**cr**' lemma [14], except for our focus on nondeterminism.

Evaluation contexts are special instances of arbitrary contexts C with any number of holes anywhere (also inside abstractions, unfoldings, etc.) In particular, actions are contexts with no holes. Contexts can be composed by filling the innermost into the outermost, $C_{outer}[C_{inner}]$, and this composition is associative. Whenever E_1 and E_2 are evaluation contexts, so is their composition, $E_1[E_2]$. Consequently, by inspection of the evaluation rules above, $A \mapsto B$ implies $E[A] \mapsto E[B]$, for all evaluation contexts E .

Remark 3.3 Notice that, given a transition $A \mapsto B$, the reduced redex need not be uniquely determined, as witnessed by the transition

$$\text{diverge and diverge} \mapsto \text{diverge and diverge} ,$$

by reduction of either of the two occurrences of redex `diverge = unfolding unfold`.

□

Recall that $_@_$ substitutes its second argument for all free occurrences of **unfold** in its first argument. With the obvious extension of $_@_$ to contexts we have, for all contexts C and actions U , that $C@U$ is a context and $C[A]@U = (C@U)[A]$, for all closed A . The definition of open/closed actions naturally generalises to contexts too. Unless otherwise stated, evaluation contexts are always assumed to be closed.

Call a context ‘finite’ if no holes occur inside the combinator **unfold**, in particular, all evaluation contexts are finite. Finite contexts F satisfy $F[A]@U = (F@U)[A@U]$, for arbitrary actions A, U . Open actions A and closed, finite contexts F are in a one-to-one correspondence, $A = F[\mathbf{unfold}]$ and $F = A@[]$, such that $A@U = F[U]$, for all U .

We are going to use some properties of the interplay between evaluation and composition, i.e. how do actions composed of a context filled with an action evaluate? Consider such an action $C[A]$ where C and A are closed. The contribution of C and A to the processing capabilities of $C[A]$ is explicated by the following lemmas which we state without proofs. For notational convenience we restrict ourselves to the case when C is a finite context F .

The first lemma says that a context either (i) has a hole at an evaluation point in the context, i.e. the context E obtained by replacing all other holes of F by **unfold** is an open evaluation context, or (ii) can evaluate without reference to what is filled into its holes.

Lemma 3.4 *For all closed, finite contexts F ,*

either (i) $(\exists \text{ open } E) F = E@[]$,

or else (ii) $(\forall \text{ closed } A, B' \mid F[A] \mapsto B')$

$(\exists \text{ closed } F' \mid B' = F'[A]) (\forall \text{ closed } A') F[A'] \mapsto F'[A']$,

where E is a (possibly open) evaluation context, and F' a finite context.

The next lemma describes the role of A in a transition $F[A] \mapsto B'$.

Lemma 3.5 *Assume F, A are closed and $F[A] \mapsto B'$. If A is not terminated,*

(i) $(\exists \text{ open } E \mid F = E@[]) (\exists \text{ closed } B \mid A \mapsto B) B' = (E@A)[B]$,

or (ii) $(\exists \text{ closed } F' \mid B' = F'[A]) (\forall \text{ closed } A') F[A'] \mapsto F'[A']$.

(i) and (ii) are not mutually exclusive, as illustrated by remark 3.3:

diverge and diverge \mapsto diverge and diverge ,

where $F = E = F' = (\mathbf{diverge \ and \ []})$, $A = B = \mathbf{diverge}$.

We shall mainly be interested in the special instance where F is an evaluation context. Then in case (i) $E = F$, and in case (ii) F' is either an evaluation context or $F' = B'$ (a context with no holes). Thus if an action makes a transition and the action decomposes into an evaluation context filled with a non-terminated subaction, then (i) the transition chooses a redex in the subaction, or a parallel redex is used which either (ii) doesn't affect the subaction, or (iii) eliminates it:

Lemma 3.6 (Choice of execution point) *If $E[A] \mapsto B'$ and A is not terminated, then*

- (i) $(\exists \text{ closed } B \mid A \mapsto B) B' = E[B]$,
- or (ii) $(\exists E' \mid B' = E'[A]) (\forall \text{ closed } A') E[A'] \mapsto E'[A']$,
- or (iii) $(\forall \text{ closed } A') E[A'] \mapsto B'$.

4 Contextual testing

In this section, we define contextual test preorders and equivalences for actions.

4.1 Definedness

Let us introduce two ‘definedness’ predicates for actions. Action A *may* succeed, $A \downarrow_{\text{may}}$, if there exists a successful computation from A . Action A *must* succeed, $A \downarrow_{\text{must}}$, if all computations from A are successful.

Definition 4.1 (Defined) \downarrow_{may} and \downarrow_{must} are the least predicates satisfying

$$\begin{aligned} A \downarrow_{\text{may}} & \text{ if } A \mapsto^* S \text{ for some } S:\text{success} . \\ A \downarrow_{\text{must}} & \text{ if } A \neq \text{fail} \text{ and } (A \mapsto B \text{ implies } B \downarrow_{\text{must}}) . \end{aligned}$$

\downarrow_{must} is defined negatively: $A \downarrow_{\text{must}}$ if no computation from A fails or diverges. (Consequently $A \downarrow_{\text{must}}$ if A is stuck, cf. Section 9).

Recall **success = complete|escape**. These are the two ‘positive’ action outcomes that can be used constructively. I.e., a context may test whether an action completes or escapes and respond accordingly,

$$C_{\text{complete}} = ([\] \text{ and then } \text{Response}) , \quad C_{\text{escape}} = ([\] \text{ trap } \text{Response}) .$$

It is impossible to respond deterministically to failed and divergent outcome.

A simple consequence of the definition of \downarrow_{may} and \downarrow_{must} is:

Fact 4.2 *If $A \mapsto B$, then $B \downarrow_{\text{may}}$ implies $A \downarrow_{\text{may}}$, and $A \downarrow_{\text{must}}$ implies $B \downarrow_{\text{must}}$.*

Let the preorders \ll_{may} and \ll_{must} order definedness of actions:

Definition 4.3 *For closed actions A_1, A_2 and $m \in \{\text{may}, \text{must}\}$,*

$$A_1 \ll_m A_2 \quad \text{iff} \quad A_1 \downarrow_m \Rightarrow A_2 \downarrow_m .$$

We will need the following inductive formulation of the definedness predicates and relations for induction proofs:

$$\begin{aligned} \downarrow_{\text{may}} &= \bigcup_{\lambda < \omega} \downarrow_{\text{may}}^\lambda \quad \text{where } \downarrow_{\text{may}}^\lambda = \text{success} \cup \\ &\quad \{ A \mid \exists B \in \bigcup_{\kappa < \lambda} \downarrow_{\text{may}}^\kappa . A \mapsto B \} . \\ \downarrow_{\text{must}} &= \bigcup_{\lambda < \omega} \downarrow_{\text{must}}^\lambda \quad \text{where } \downarrow_{\text{must}}^\lambda = \{ A \mid A \neq \text{fail} \wedge \\ &\quad (\forall B \mid A \mapsto B) B \in \bigcup_{\kappa < \lambda} \downarrow_{\text{must}}^\kappa \} . \\ \ll_m &= \bigcap_{\lambda < \omega} \ll_m^\lambda \quad \text{where } A_1 \ll_m^\lambda A_2 \Leftrightarrow A_1 \downarrow_m^\lambda \Rightarrow A_2 \downarrow_m . \end{aligned}$$

If countable nondeterminism is added, \downarrow_{must} and \ll_{must} run over all recursive ordinals as in definition 3.2. We shall phrase all induction proofs so that they also hold for transfinite induction and thus carry over to the more general case.

4.2 Contextual testing equivalence

Now we can define three contextual testing equivalences. They are derived from **may** and **must** preorders defined as the weakest precongruences ordering divergent actions below convergent ones, i.e. included in \ll_{may} and \ll_{must} .

Definition 4.4 (Contextual preorders) *For arbitrary actions A_1, A_2 ,*

$$\begin{aligned} A_1 \sqsubseteq_{\text{may}} A_2 &\quad \text{iff} \quad \forall \text{ closing contexts } C. C[A_1] \ll_{\text{may}} C[A_2] , \\ A_1 \sqsubseteq_{\text{must}} A_2 &\quad \text{iff} \quad \forall \text{ closing contexts } C. C[A_1] \ll_{\text{must}} C[A_2] , \end{aligned}$$

where C is closing if $C[A_i]$ is closed, for $i = 1, 2$.

These contextual testing preorders generalise conventional contextual preorder to the nondeterministic case. Notice that these preorders are defined for both open and closed actions. We shall focus on the ‘closed’ preorders, i.e. these preorders restricted to closed actions only. (Section 5.3 will explain how to generalise our results to the general case.)

Let \simeq_{may} and \simeq_{must} denote the induced equivalences. Define ‘contextual equivalence’ as $\simeq = \simeq_{\text{may}} \cap \simeq_{\text{must}}$.

These equivalences are, by definition, the weakest equivalences such that, in any context, equivalent subterms are interchangeable without any observable effect. One chooses **may**, **must**, or their intersection ‘ \simeq ’, dependent on one’s view of observation: whether one ignores divergence, regards divergence as catastrophic, or demands equal divergence properties, respectively.

As a rough intuition of the \sqsubseteq_{may} and $\sqsubseteq_{\text{must}}$ preorders, they both mean ‘the left argument is more divergent than the right argument’, and both \sqsupseteq_{may} and

$\sqsubseteq_{\text{must}}$ mean ‘left argument is less deterministic than right argument’. Thus the divergent action **diverge** is both \perp_{may} and \perp_{must} , and the completely unpredictable action **chaos** is both \top_{may} and \perp_{must} . Refer to the latter as the ‘implementation’ preorder, $\sqsubseteq_{\text{impl}} = \sqsupseteq_{\text{may}} \cap \sqsubseteq_{\text{must}}$, i.e. specification S is implemented by the more deterministic implementation I if $S \sqsubseteq_{\text{impl}} I$. Notice that $\simeq = \sqsubseteq_{\text{impl}} \cap \sqsupseteq_{\text{impl}}$.

Having all the contextual preorders, we can express more properties of actions by a variety of inequational laws. This will be exemplified in Section 8.

\sqsubseteq_{may} and $\sqsubseteq_{\text{must}}$ can be viewed as testing preorders, in the sense of [13], by taking the set of ‘tests’ or ‘experimenters’ to be all action contexts. A test or experiment C is performed on an action A by performing $C[A]$. If the computation is successful, A ‘passes’ test C .

The testing equivalence defined for actions in [22, C.4] corresponds to our **may** equivalence. (Both the notion of test and the notion of observation used there are a little less standard.) We shall instead take contextual equivalence, \simeq , as basic. It is a stronger equivalence, and thus a ‘safe compromise’ between the **may** and the **must** view of divergence, and it turns out that the action laws hold even for contextual equivalence.

Remark 4.5 (Reactive formulation) We would like our theory to generalise to AN’s communicative facet also. Testing for some reactive behaviour would seem more appropriate in that setting. In [13] a special action w signals success. Similarly, the test setup in [1] has an observer primitive event(). Suppose we define a special action **event**, that may occur in test contexts only, and check whether **event** may or must be performed during testing, then the resulting preorders will be the same as in definition 4.4. It is easy to see that the new preorders are as discriminating as the old: just replace every old context C_{success} by

$$C_{\text{event}} = (C_{\text{success}} \text{ and then event}) \text{ trap event} .$$

The opposite is less direct: in every new context, replace occurrences of **event** by **escape** and ensure that this exception is not trapped (somehow resolve conflicts with traps and other escapes in the context). \square

4.3 Basic action theory

There are, in fact, only 4 **may** and 8 **must** equivalence classes for closed basic actions. Define **diverge**, **succeed**, **terminate**, and **chaos**.

$$\begin{aligned} \text{diverge} &= \text{unfolding unfold} . \\ \text{succeed} &= \text{complete or escape} . \\ \text{terminate} &= \text{succeed trap (escape and fail)} . \end{aligned}$$

$$\text{chaos} = (\text{diverge and fail and escape}) \text{ trap succeed} .$$

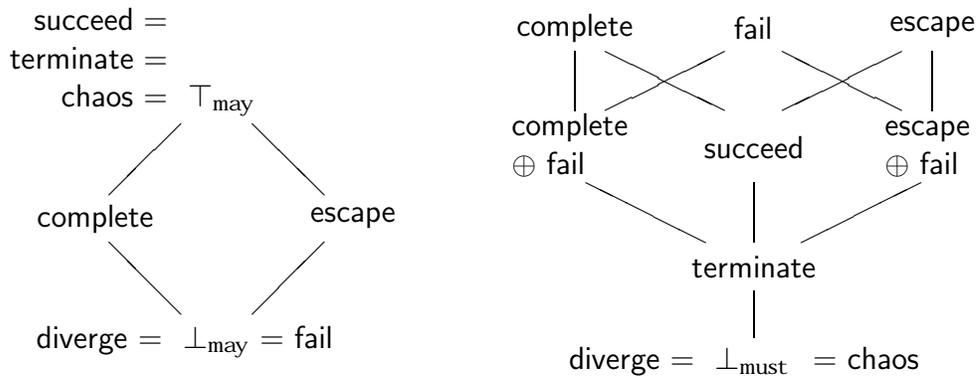
The **or** combinator discards any failing argument, i.e. if one argument fails the other is chosen. This gives **or** an ‘angelic’ flavour and makes it inapplicable for straightforward formulations of some of the actions above. To further the intuition behind the latter three, suppose we had a binary nondeterministic ‘internal’ or ‘demonic’ choice operator \oplus (as in [13]) which chooses any of its two arguments without inspection of the other, then

$$\begin{aligned} \text{succeed} &= \text{complete} \oplus \text{escape} . \\ \text{terminate} &= \text{success} \oplus \text{fail} . \\ \text{chaos} &= \text{terminate} \oplus \text{diverge} . \end{aligned}$$

$\text{complete} \oplus \text{fail}$, $\text{escape} \oplus \text{fail}$ can be coded as

$$\begin{aligned} \text{complete} \oplus \text{fail} &= \text{succeed trap fail} . \\ \text{escape} \oplus \text{fail} &= \text{succeed and then fail} . \end{aligned}$$

The closed **may** and **must** orderings can be depicted as follows:²



(Notice that \oplus is the **may** least upper bound operator, ‘ \sqcup_{may} ’, and the **must** greatest lower bound, ‘ \sqcap_{must} ’, hence also the **impl** greatest lower bound, ‘ \sqcap_{impl} ’.)

In Section 7 we will show that the diagrams exactly describe the closed preorders. Here we only show that the **may** and **must** preorders are at least as discriminative as illustrated by the diagrams. To this end we define simple preorders that are in exact correspondence with the diagrams.

Definition 4.6 For closed actions A_1, A_2 , define

$$\begin{aligned} A_1 \simeq_{\text{may}} A_2 &\text{ iff } (\forall S:\text{success} \mid A_1 \mapsto^* S) A_2 \mapsto^* S , \\ A_1 \simeq_{\text{must}} A_2 &\text{ iff } A_1 \Downarrow \Rightarrow A_2 \Downarrow \\ &\quad \wedge (\forall T:\text{terminated} \mid A_2 \mapsto^* T) A_1 \mapsto^* T . \end{aligned}$$

²In [17] the **must** picture was erroneous.

These relations are easily shown to be preorders.

Proposition 4.7 *The diagrams are exact descriptions of \simeq_{may} and \simeq_{must} .*

Proof (sketch): By inspection of the diagrams, the preorders can be seen to make all the (implicit) distinctions in the diagrams: no two nodes in the diagrams can be merged and no lines can be added between the nodes.

Conversely, simple combinatorial arguments based on the definitions of \simeq_{may} and \simeq_{must} show that there are no more \simeq equivalence classes than illustrated in the diagrams (in the **must** case we exclude the stuck equivalence class, cf. Section 9). It is easily checked that all lines between nodes reflect the preorders. \square

We shall now prove that the closed contextual preorders are included in \simeq_{may} and \simeq_{must} . Thus it follows by proposition 4.7 that the contextual preorders are at least as discriminative as depicted by the diagrams.

Lemma 4.8 *For closed actions A_1, A_2 ,*

$$\begin{aligned} A_1 \sqsubseteq_{\text{may}} A_2 & \text{ implies } A_1 \simeq_{\text{may}} A_2 . \\ A_1 \sqsubseteq_{\text{must}} A_2 & \text{ implies } A_1 \simeq_{\text{must}} A_2 . \end{aligned}$$

Proof: Let $m \in \{\text{may}, \text{must}\}$. Assume $A_1 \sqsubseteq_m A_2$, hence $C[A_1] \downarrow_m \Rightarrow C[A_2] \downarrow_m$, for all closed C . Show $A_1 \simeq_m A_2$:

$m = \text{may}$: Let $C_1 = [] \text{ trap fail}$, $C_2 = [] \text{ and then fail}$. Examination of the evaluation relation shows, for all closed A ,

$$C_i[A] \downarrow_{\text{may}} \Leftrightarrow A \mapsto^* S_i, \text{ for } i = 1, 2,$$

where $S_1 = \text{complete}$, $S_2 = \text{escape}$. Thus $C_i[A_1] \downarrow_{\text{may}} \Rightarrow C_i[A_2] \downarrow_{\text{may}}$ gives $A_1 \mapsto^* S_i \Rightarrow A_2 \mapsto^* S_i$, for $i = 1, 2$. This is exactly the definition of $A_1 \simeq_{\text{may}} A_2$.

$m = \text{must}$: Let $C_0 = [] \text{ or complete}$,
 $C_1 = ([] \text{ or escape}) \text{ and then fail}$,
 $C_2 = ([] \text{ or complete}) \text{ trap fail}$,
 $C_3 = []$.

These contexts satisfy, for all closed A ,

$$\begin{aligned} C_0[A] \downarrow_{\text{must}} & \Leftrightarrow A \Downarrow, \\ C_i[A] \downarrow_{\text{must}} & \Leftrightarrow A \Downarrow \wedge A \not\mapsto^* T_i, \text{ for } i = 1, \dots, 3, \end{aligned}$$

where $T_1 = \text{complete}$, $T_2 = \text{escape}$, $T_3 = \text{fail}$. Expand $C_i[A_1] \downarrow_{\text{must}} \Rightarrow C_i[A_2] \downarrow_{\text{must}}$, for $i = 0, \dots, 3$, according to the above. Putting it all together yields the definition of $A_1 \simeq_{\text{must}} A_2$. \square

5 Finite testing

The contextual test preorders are defined by quantification over arbitrary action contexts which makes it hard to reason about the preorders directly from the definition. In Section 6 we show that it suffices to quantify over evaluation contexts (the **basic context lemma** 6.10) and use this to show actions preordered. It is advantageous to perform the restriction from arbitrary contexts to evaluation contexts in two steps, namely first to restrict to finite contexts (where no holes occur inside the combinator **unfolding**). The restriction to finite contexts is most conveniently proved separately because it involves a thorough treatment of open and closed actions and contexts. The further restriction to evaluation contexts need only deal with closed actions and thus becomes more clear.

Before proceeding to Section 6 we carry out this first step. We shall prove the following lemma:

Lemma 5.1 (Finite restriction) *For closed A_1, A_2 and $m \in \{\text{may}, \text{must}\}$,*

$$A_1 \sqsubseteq_m A_2 \quad \text{if and only if} \quad \forall \text{ closed, finite contexts } F. F[A_1] \ll_m F[A_2].$$

The ‘only if’ direction is immediate because closed, finite contexts are instances of arbitrary closed contexts. (Observe that for closed actions, every closed context is a closing context.) We shall postpone the proof of the ‘if’ direction till Section 5.2.

5.1 Open extension

The contextual preorders \sqsubseteq_m and equivalences \simeq_m are defined for arbitrary actions. Yet, Section 4.3 as well as the following sections deal only with closed actions. In particular, we shall develop alternative characterisations of the semantic preorders that will be defined for closed actions only. In the terminology of Gordon [10], define their ‘open extensions’ as follows:

Given a relation on closed actions, $\mathcal{R} \subseteq \text{closed-action} \times \text{closed-action}$, its *open extension* is a relation on open actions $\mathcal{R}^\circ \subseteq \text{action} \times \text{action}$,

$$A_1 \mathcal{R}^\circ A_2 \stackrel{\text{def}}{\iff} \forall U:\text{closed-action}. (A_1 @ U) \mathcal{R} (A_2 @ U),$$

for arbitrary actions A_1, A_2 . The open extensions of relations \mathcal{R} and \mathcal{S} coincide, $\mathcal{R}^\circ = \mathcal{S}^\circ$, if and only if the restrictions to closed actions of \mathcal{R} and \mathcal{S} coincide.

The proof of lemma 5.1 will establish that \sqsubseteq_m° is contained in \sqsubseteq_m , see Section 5.3. Hence any safe approximation \mathcal{R} to \sqsubseteq_m on closed actions yields a safe approximation \mathcal{R}° to \sqsubseteq_m . Therefore we shall be content with developing characterisations of \sqsubseteq_m on closed actions.

5.2 Precongruence

Lemma 5.1 can be restated as $\sqsubset_m^\circ = \lesssim_m^\circ$, for $m \in \{\mathbf{may}, \mathbf{must}\}$, where:

Definition 5.2 For all closed A_1, A_2 ,

$$A_1 \lesssim_m A_2 \stackrel{\text{def}}{\iff} \forall \text{ closed, finite } F. F[A_1] \ll_m F[A_2] .$$

Because of the correspondence between closed, finite contexts and open actions,

Fact 5.3 $A_1 \lesssim_m A_2 \iff \forall A. A@A_1 \ll_m A@A_2$.

Our proof of lemma 5.1 follows the approach of Mason, Smith, and Talcott [14]. The crux of the proof consists of showing that \lesssim_m° is a precongruence.

Proposition 5.4 \lesssim_m° is a precongruence, i.e.

$$A_1 \lesssim_m^\circ A_2 \quad \text{implies} \quad C[A_1] \lesssim_m^\circ C[A_2] ,$$

for arbitrary actions A_1, A_2 , contexts C , and $m \in \{\mathbf{may}, \mathbf{must}\}$.

Proof: By structural induction on C .

If $C = []$ or C has no holes, the result is immediate, in the latter case because \lesssim_m° is reflexive.

If $C = O(C_1 \dots C_n)$ and $O \neq \mathbf{unfolding}$, then, for all closed, finite contexts F and closed actions U ,

$$\begin{aligned} F[C[A_1]@U] &= F[O(C_1 \dots C_n)[A_1]]@U , \text{ since } F \text{ is closed and finite} \\ &= F_1[C_1[A_1]@U] , \text{ where } F_1 = F[O([] C_2[A_1] \dots C_n[A_1])]@U \\ &\ll_m F_1[C_1[A_2]@U] , \text{ since } C_1[A_1] \lesssim_m^\circ C_1[A_2] \text{ by I.H.} \\ &\quad \text{and } F_1 \text{ is closed and finite} \\ &= F_2[C_2[A_1]@U] \\ &\quad \vdots \\ &= F_n[C_n[A_1]@U] \\ &\ll_m F_n[C_n[A_2]@U] , \text{ by I.H. since } F_n \text{ is closed and finite} \\ &= F[C[A_2]@U] , \end{aligned}$$

where $F_i = F[O(C_1[A_2] \dots C_{i-1}[A_2] [] C_{i+1}[A_1] \dots C_n[A_1])]@U$. Conclude $C[A_1] \lesssim_m^\circ C[A_2]$.

If $C = \mathbf{unfolding} C_0$, let $A'_i = C_0[A_i]$, $U_i = C[A_i] = \mathbf{unfolding} A'_i$, for $i = 1, 2$. $A'_1 \lesssim_m^\circ A'_2$ by I.H. We must show $U_1 \lesssim_m^\circ U_2$ which is equivalent to $U_1 \lesssim_m U_2$ because U_i is closed. We show $\forall \lambda < \omega. P(\lambda)$, where $P(\lambda) \iff \forall \text{ closed } F. F[U_1] \ll_m^\lambda F[U_2]$, by induction on λ . Let $\lambda < \omega$, assume $\forall \kappa < \lambda. P(\kappa)$ and show $P(\lambda)$:

$m = \mathbf{may}$: Assume $F[U_1] \downarrow_{\mathbf{may}}^\lambda$.

Either (1) $F[U_1] : \mathbf{success}$, then $F : \mathbf{success}$, hence $F[U_2] : \mathbf{success}$ and $F[U_2] \downarrow_{\mathbf{may}}$.

Or (2) $F[U_1] \mapsto B'$, $B' \downarrow_{\mathbf{may}}^\kappa$ for some $\kappa < \lambda$. By lemma 3.5:

Either (i) $B' = (E@U_1)[B]$ where $F = E@[]$ and $U_1 \mapsto B$. Then $B = A'_1@U_1$ by evaluation rule (3). Observe $B' = E[A'_1]@U_1$. By I.H. and fact 5.3, $B' \downarrow_{\text{may}}^\kappa$ implies $E[A'_1]@U_2 \downarrow_{\text{may}}$ which can be written $(E@U_2)[A'_1@U_2] \downarrow_{\text{may}}$. This implies $(E@U_2)[A'_2@U_2] \downarrow_{\text{may}}$ since $A'_1 \lesssim_{\text{may}}^\circ A'_2$. It is easily checked that $F[U_2] \mapsto (E@U_2)[A'_2@U_2]$, hence $F[U_2] \downarrow_{\text{may}}$ by fact 4.2.

Or (ii) $B' = F'[U_1]$ and $F[U_2] \mapsto F'[U_2]$. Since $F'[U_1] \downarrow_{\text{may}}^\kappa$, $F'[U_2] \downarrow_{\text{may}}$ by I.H., hence $F[U_2] \downarrow_{\text{may}}$ by fact 4.2.

$m = \text{must}$: Assume $F[U_1] \downarrow_{\text{must}}^\lambda$.

$F[U_2] = \text{fail}$ only if $F = \text{fail}$, in which case $F[U_1] = \text{fail}$. So $F[U_1] \neq \text{fail}$ implies $F[U_2] \neq \text{fail}$. What remains to be shown is whenever $F[U_2] \mapsto B'$, $B' \downarrow_{\text{must}}$. By lemma 3.5:

Either (i) $B' = (E@U_2)[B]$ where $F = E@[]$, $U_2 \mapsto B$, and $B = A'_2@U_2$. Observe $B' = E[A'_2]@U_2$ and $F[U_1] \mapsto E[A'_1]@U_1$. Hence $E[A'_1]@U_1 \downarrow_{\text{must}}^\kappa$ for some $\kappa < \lambda$. By I.H. and fact 5.3, $E[A'_1]@U_2 \downarrow_{\text{must}}$. Now $E[A'_2]@U_2 \downarrow_{\text{must}}$ follows from $A'_1 \lesssim_{\text{must}}^\circ A'_2$, i.e. $B' \downarrow_{\text{must}}$.

Or (ii) $B' = F'[U_2]$ and $F[U_1] \mapsto F'[U_1]$. Hence $F[U_1] \downarrow_{\text{must}}^\kappa$ for some $\kappa < \lambda$. By I.H. $F[U_2] \downarrow_{\text{must}}$, i.e. $B' \downarrow_{\text{must}}$.

Thus $\forall \lambda < \omega. P(\lambda)$ is established which concludes the proof. \square

We are now in a position to prove the ‘if’ direction of lemma 5.1. Since $\lesssim_{@m}^\circ$ is a precongruence and contained in \ll_m (trivial from the definition of $\lesssim_{@m}^\circ$), $\lesssim_{@m}^\circ \subseteq \sqsubset_m$, because \sqsubset_m is defined as the weakest precongruence contained in \ll_m . In particular, this is the case when restricted to closed actions, $\lesssim_{@m} \subseteq \sqsubset_m$. More formally:

Proof (lemma 5.1, ‘if’): Let $m \in \{\text{may}, \text{must}\}$ and A_1, A_2 be closed actions. Assume $A_1 \lesssim_{@m} A_2$, hence $A_1 \lesssim_{@m}^\circ A_2$, and let C be any closing context and show $C[A_1] \ll_m C[A_2]$. C is closed so by the precongruence of $\lesssim_{@m}^\circ$ (proposition 5.4), $C[A_1] \lesssim_{@m}^\circ C[A_2]$. Conclude since $\lesssim_{@m}^\circ$ is included in \ll_m . \square

Thus $\lesssim_{@m}$ and \sqsubset_m coincide for closed actions and consequently $\lesssim_{@m}^\circ$ and \sqsubset_m° coincide for arbitrary actions since the open extension \mathcal{R}° of a relation \mathcal{R} by definition is completely determined by the restriction of \mathcal{R} to closed actions. Hence proposition 5.4 implies that \sqsubset_m° is a precongruence.

5.3 Soundness of open extension

Analogously to the proof of lemma 5.1 we get that \sqsubset_m° , being a precongruence, is included in \sqsubset_m :

Lemma 5.5 For arbitrary actions A_1, A_2 ,

$$A_1 \overset{\circ}{\simeq}_m A_2 \quad \text{implies} \quad A_1 \simeq_m A_2 .$$

Proof: Assume $A_1 \overset{\circ}{\simeq}_m A_2$ and C is any closing context. By the precongruence of $\overset{\circ}{\simeq}_m$, $C[A_1] \overset{\circ}{\simeq}_m C[A_2]$. Since $C[A_i]$ is closed, $C[A_1] \simeq_m C[A_2]$. Hence $C[A_1] \ll_m C[A_2]$ because $\overset{\circ}{\simeq}_m$ by definition is included in \ll_m . \square

This is the most important direction of implication. It means that for showing open actions preordered, it suffices to show them preordered by the open extension. In many common cases results about closed preorderings generalise straightforwardly to their open extensions.

E.g., for closed actions A_1, A_2 , $A_1 \mathcal{R}^\circ A_2 \Leftrightarrow A_1 \mathcal{R} A_2$. More generally:

Fact 5.6 Let \mathcal{R} be a relation on closed actions. For a clause of the form

$$\forall A_1..A_n. LHS(A_1, \dots, A_n) \mathcal{R}^\circ RHS(A_1, \dots, A_n) , \quad (1)$$

(normally the universal quantification over arbitrary actions is left implicit in action laws), in two cases it suffices to show that it holds for the closed relation \mathcal{R} .

- (i) When LHS and RHS are both ‘closing’, i.e., for arbitrary actions A_1, \dots, A_n , $LHS(A_1, \dots, A_n)$ and $RHS(A_1, \dots, A_n)$ are closed. In particular, this is the case when $n = 0$ and $LHS()$ and $RHS()$ are closed actions.
- (ii) When $@$ distributes over both LHS and RHS, i.e.

$$\begin{aligned} LHS(A_1, \dots, A_n) @ U &= LHS(A_1 @ U, \dots, A_n @ U) , \\ RHS(A_1, \dots, A_n) @ U &= RHS(A_1 @ U, \dots, A_n @ U) , \end{aligned} \quad (2)$$

for arbitrary A_1, \dots, A_n and closed U , then (1) is equivalent to

$$\forall \text{closed } A'_1..A'_n. LHS(A'_1, \dots, A'_n) \mathcal{R} RHS(A'_1, \dots, A'_n) . \quad (3)$$

To see this, observe that, by definition of \mathcal{R}° , (1) holds iff

$$\forall A_1..A_n. \forall \text{closed } U. LHS(A_1, \dots, A_n) @ U \mathcal{R} RHS(A_1, \dots, A_n) @ U . \quad (4)$$

This is equivalent to (3) by property (2) and because $A'_i = A_i @ U$ is closed when U is and $A'_i @ U = A'_i$ when A'_i is closed.

With regard to (ii), note that $@$ distributes over all action constructs other than unfolding .

For instance, we only need to prove that the action laws from Section 2.4 hold for \simeq for closed actions (see Section 8): All the laws generalise to arbitrary actions and \simeq° according to fact 5.6, hence also hold for \simeq by lemma 5.5.

The converse of lemma 5.5 is not true for basic actions only, at least not in the **must** case:

Example 5.7 Let $A_1 = (\text{unfold and then complete})$, $A_2 = (\text{unfold and then escape})$. They are distinguished by \sqsubset_m° by means of context $C = ([] \text{ trap fail})$ and closed action $U = \text{complete}$,

$$\begin{aligned} C[A_1@U] &= (\text{complete and then complete}) \text{ trap fail} \simeq \text{complete} , \\ C[A_2@U] &= (\text{complete and then escape}) \text{ trap fail} \simeq \text{fail} . \end{aligned}$$

Hence $A_1 \not\approx_{\text{must}}^\circ A_2$.

When a closing context C' is fed with an action A , either $C'[A]$ can only perform without reference to A (A is ‘dead code’), or else performance of $C'[A]$ can reach A . In the latter case, due to the simplicity of basic actions, if performance of A can reach an occurrence of **unfold** in A , A can necessarily be reached again, hence $C'[A]$ may diverge. Consequently, either C' doesn’t distinguish actions at all, or else, for all actions A which can reach an occurrence of **unfold**, $C'[A]$ is equated to $\perp_{\text{must}} = \text{diverge}$ in the **must** ordering. For the example above, since A_1 and A_2 both have a reachable occurrence of **unfold**, $A_1 \simeq_{\text{must}} A_2$. \square

When other facets are added, action contexts become more expressive and the converse does hold. (For every closed context C and closed action U , it is possible to produce a closing context C_U such that $C[A@U] \simeq C_U[A]$, for all A .)

6 Alternative characterisation

The definition of contextual preorders is difficult to use to prove actions pre-ordered or equivalent. We need an alternative characterisation which gives a tractable way of determining equivalence. This approach is used by Hennessy for test preorders [13] and it is central to Mason and Talcott’s operational work. Inspired by the ‘ciu’ characterisation of the latter [19], define the following preorders for actions. Following Gordon [11], we call them ‘experimental’ preorders.

Definition 6.1 (Experimental preorders) For closed actions A_1, A_2 ,

$$\begin{aligned} A_1 \lesssim_{\text{may}} A_2 &\quad \text{iff} \quad \forall E. E[A_1] \ll_{\text{may}} E[A_2] , \\ A_1 \lesssim_{\text{must}} A_2 &\quad \text{iff} \quad \forall E. E[A_1] \ll_{\text{must}} E[A_2] , \end{aligned}$$

where E ranges over closed evaluation contexts.

‘Experimental equivalence’, \approx , denotes the intersection of the two induced equivalences. Let $\lesssim_{\text{impl}} = \lesssim_{\text{may}} \cap \lesssim_{\text{must}}$, then $\approx = \lesssim_{\text{impl}} \cap \gtrsim_{\text{impl}}$.

These preorders only quantify over evaluation contexts, not arbitrary contexts. This turns out to be much more tractable.

Just as for the inductive formulation of \ll_{may} and \ll_{must} , it is possible to express the experimental preorders as limits of sequences of approximations. For $m \in \{\text{may}, \text{must}\}$,

$$\lesssim_m = \bigcap_{\lambda < \omega} \lesssim_m^\lambda \quad \text{where } A_1 \lesssim_m^\lambda A_2 \Leftrightarrow \forall E. E[A_1] \ll_m^\lambda E[A_2] ,$$

This gives a useful proof technique for showing actions experimental preordered: Prove $\forall \lambda < \omega. A_1 \lesssim_m^\lambda A_2$ by induction on λ .

6.1 Experimental order proof technique

The beauty of the experimental preorders is that actions can be proven experimental preordered essentially by induction on the length of their computations. The possibly interleaved performance of actions adds some complications (compared to Mason and Talcott's sequential setting [14]; their parallel work [1] is not directly comparable to the parallelism of basic actions).

As an illustration of the proof technique employed, we shall prove two 'evaluation lemmas'.

First the more general and weaker version which is proved by reference to fact 4.2:

Lemma 6.2 (Evaluation) $A_1 \mapsto A_2$ implies $A_1 \lesssim_{\text{impl}} A_2$.

Proof: If $A_1 \mapsto A_2$, then also $E[A_1] \mapsto E[A_2]$, for all evaluation contexts E .

First show $A_1 \gtrsim_{\text{may}} A_2$, i.e. $\forall E. E[A_2] \downarrow_{\text{may}} \Rightarrow E[A_1] \downarrow_{\text{may}}$. If $E[A_2] \downarrow_{\text{may}}$, then since $E[A_1] \mapsto E[A_2]$, $E[A_1] \downarrow_{\text{may}}$, by fact 4.2.

Next show $A_1 \lesssim_{\text{must}} A_2$, i.e. $\forall E. E[A_1] \downarrow_{\text{must}} \Rightarrow E[A_2] \downarrow_{\text{must}}$. If $E[A_1] \downarrow_{\text{must}}$, then since $E[A_1] \mapsto E[A_2]$, $E[A_2] \downarrow_{\text{must}}$, by fact 4.2. \square

In accordance with our intuition that \lesssim_{impl} means 'less deterministic than', we may read the preceding evaluation lemma as 'evaluation reduces nondeterminism'.

Corollary 6.3 *Some immediate consequences are:*

- (i) complete or $A \lesssim_{\text{impl}} \text{complete}$.
- (ii) escape or $A \lesssim_{\text{impl}} \text{escape}$.
- (iii) fail or $A \lesssim_{\text{impl}} A$.
- (iv) complete and $A \lesssim_{\text{impl}} A$.
- (v) escape and $A \lesssim_{\text{impl}} \text{escape}$.

(vi) **fail** and $A \lesssim_{\text{impl}} \text{fail}$.

In deterministic settings, when one term can make a transition into another, then the terms are equivalent [9, 19]. This demonstrates how the presence of nondeterminism weakens our theory. Under certain conditions we may strengthen the evaluation lemma, however.

The proof of the following ‘determined’ version of the evaluation lemma employs the typical machinery for proofs of experimental orderings, namely induction on the length of computation, λ , using lemma **choice of execution point** (3.6).

Lemma 6.4 (Determined evaluation)

If $A_1 \mapsto A_2$ is the only transition out of A_1 , then $A_1 \approx A_2$.

Proof: $A_1 \gtrsim_{\text{may}} A_2$ and $A_1 \lesssim_{\text{must}} A_2$ follow from lemma 6.2. Note that since $A_1 \mapsto A_2$, A_1 is not terminated.

$A_1 \lesssim_{\text{may}} A_2$ follows from $\forall \lambda < \omega. A_1 \lesssim_{\text{may}}^\lambda A_2$. Show this by induction on λ . Let $\lambda < \omega$, assume $\forall \kappa < \lambda. A_1 \lesssim_{\text{may}}^\kappa A_2$, and show $A_1 \lesssim_{\text{may}}^\lambda A_2$: Assume $E[A_1] \downarrow_{\text{may}}^\lambda$. We know A_1 is not terminated, so $E[A_1]$ is not in **success**. By the definition of $\downarrow_{\text{may}}^\lambda$, there is a B' such that $E[A_1] \mapsto B'$ and $B' \downarrow_{\text{may}}^\kappa$, for some $\kappa < \lambda$. Show $E[A_2] \downarrow_{\text{may}}$ by application of lemma **choice of execution point** to $E[A_1] \mapsto B'$:

Either (i) $B' = E[B]$ where $A_1 \mapsto B$. Since $A_1 \mapsto A_2$ is the only transition out of A_1 , $B = A_2$, $B' = E[A_2]$, and $E[A_2] \downarrow_{\text{may}}^\kappa$, hence $E[A_2] \downarrow_{\text{may}}$.

Or (ii) $B' = E'[A_1]$ and, for all A' , $E[A'] \mapsto E'[A']$. In particular, $E[A_2] \mapsto E'[A_2]$. Since $E'[A_1] \downarrow_{\text{may}}^\kappa$ we have $E'[A_2] \downarrow_{\text{may}}$ by I.H., hence $E[A_2] \downarrow_{\text{may}}$ by fact 4.2.

Or (iii) $E[A'] \mapsto B'$ for all A' . In particular, $E[A_2] \mapsto B'$, since $B' \downarrow_{\text{may}}$ we have $E[A_2] \downarrow_{\text{may}}$ by fact 4.2.

This concludes $A_1 \lesssim_{\text{may}} A_2$.

To show $A_1 \gtrsim_{\text{must}} A_2$, show $\forall \lambda < \omega. A_2 \lesssim_{\text{must}}^\lambda A_1$ by induction on λ . Let $\lambda < \omega$, assume $\forall \kappa < \lambda. A_2 \lesssim_{\text{must}}^\kappa A_1$, and show $A_2 \lesssim_{\text{must}}^\lambda A_1$: Assume $E[A_2] \downarrow_{\text{must}}^\lambda$. Since A_1 is not terminated, $E[A_1] \neq \text{fail}$. What remains to be shown is that $E[A_1] \mapsto B'$ implies $B' \downarrow_{\text{must}}$, for all B' . By lemma **choice of execution point**:

Either (i) $B' = E[B]$ where $A_1 \mapsto B$. Since $A_1 \mapsto A_2$ is the only transition out of A_1 , $B = A_2$, $B' = E[A_2]$, and $B' \downarrow_{\text{must}}^\kappa$, hence $B' \downarrow_{\text{must}}$.

Or (ii) $B' = E'[A_1]$ and, for all A' , $E[A'] \mapsto E'[A']$. In particular, $E[A_2] \mapsto E'[A_2]$. $E[A_2] \downarrow_{\text{must}}^\lambda$ implies $E'[A_2] \downarrow_{\text{must}}^\kappa$ for some $\kappa < \lambda$. By I.H. $E'[A_1] \downarrow_{\text{must}}$, hence $B' \downarrow_{\text{must}}$.

Or (iii) $E[A'] \mapsto B'$ for all A' . In particular, $E[A_2] \mapsto B'$, hence $B' \downarrow_{\text{must}}$ by fact 4.2.

Conclude $A_1 \approx_{\text{must}} A_2$. □

Corollary 6.5 *Some laws from Section 2.4 are direct consequences:*

- (1) check true \approx complete ; check false \approx fail .
- (2) unfolding $A \approx A @$ unfolding A .
- (4)a complete and then $A \approx A$.
- (5) escape and then $A \approx$ escape ; fail and then $A \approx$ fail .
- (7)a escape trap $A \approx A$.
- (8) complete trap $A \approx$ complete ; fail trap $A \approx$ fail .

Finally, we prove that fail is the least element of \approx_{may} , a fact that will be used in Section 7. The proof is somewhat involved.

Proposition 6.6 fail $\approx_{\text{may}} A$, for all closed A .

Proof: Simultaneously, we prove the inequational law $A_1, A_2 \approx_{\text{may}} (A_1 \text{ or } A_2)$. Show $\forall \lambda < \omega. P(\lambda)$, where

$$P(\lambda) \Leftrightarrow \forall \text{ closed } A, A', E. \quad \begin{array}{l} \text{(i)} \quad E[\text{fail}] \downarrow_{\text{may}}^\lambda \Rightarrow E[A] \downarrow_{\text{may}} \\ \wedge \quad \text{(ii)} \quad E[A] \downarrow_{\text{may}}^\lambda \vee E[A'] \downarrow_{\text{may}}^\lambda \Rightarrow E[A \text{ or } A'] \downarrow_{\text{may}}, \end{array}$$

by induction on λ . Let $\lambda < \omega$, assume $\forall \kappa < \lambda. P(\kappa)$ and show $P(\lambda)$:

First (i): Clearly $E \neq []$ since not fail \downarrow_{may} . So let $E = E_0[E_1]$ where E_1 is the innermost layer of E ; either $E_1 = ([\text{seq } A_1])$, $E_1 = ([\text{sym } A_1])$, or $E_1 = (A_1 \text{ sym } [])$, (cf. the definition of evaluation contexts in Section 3.1).

Clearly $E[\text{fail}]$ is not in success. Therefore there must be a transition $E[\text{fail}] \mapsto B'$ such that $B' \downarrow_{\text{may}}^\kappa$ for some $\kappa < \lambda$. We will show that $E[A] \mapsto B''$ or $E[A] = B''$ where

- (*) $B'' = B'$,
- or (**) $B' = E'[\text{fail}]$ and $B'' = E'[A'']$ for some A', E' ,
- or (***) $B' = E'[A_1]$ and either $B'' = E'[A \text{ or } A_1]$ or $B'' = E'[A_1 \text{ or } A]$.

In any case, conclude $B'' \downarrow_{\text{may}}$; in case (*) because $B' \downarrow_{\text{may}}^\kappa$; in case (**) by I.H. (i); in case (***) by I.H. (ii). Therefore $E[A] \downarrow_{\text{may}}$ by fact 4.2.

We have $E_0[E_1[\text{fail}]] = E[\text{fail}] \mapsto B'$ and $E_1[\text{fail}]$ is not terminated so by lemma **choice of execution point**:

Either (i) $B' = E_0[B]$ and $E_1[\text{fail}] \mapsto B$. Tedious examination of the evaluation rules shows either

- (a) $B = \text{fail}$, then conclude (**) with $E'' = E_0$, $A'' = E_1[A]$; or
- (b) $B = A_1$ and $E_1[A] \mapsto A_1$, then (*); or

- (c) $B = E'_1[\text{fail}]$ where $E'_1 = ([] \text{ sym } A'_1)$ or $E'_1 = (A'_1 \text{ sym } [])$ and $A_1 \mapsto A'_1$, then $(**)$ with $E' = E_0[E'_1]$ and $A'' = A$; or
- (d) $B = A_1$ and $\text{sym} = \text{or}$, then $(***)$ with $E' = E_0$.

Or (ii) $B' = E''[E_1[\text{fail}]]$ and $E_0[E_1[A]] \mapsto E''[E_1[A]]$. Then $(**)$ holds with $E' = E''[E_1]$ and $A'' = A$.

Or (iii) $E_0[E_1[A]] \mapsto B'$, thus $(*)$.

Secondly (ii): Assume $E[A] \downarrow_{\text{may}}^\lambda$ (the case where only $E[A'] \downarrow_{\text{may}}^\lambda$ is symmetrical). If $A = \text{fail}$, then $E[A \text{ or } A'] \downarrow_{\text{may}}$ by (i) above. If $A:\text{success}$, evaluation rule (7) gives $E[A \text{ or } A'] \mapsto E[A]$ and conclude $E[A \text{ or } A'] \downarrow_{\text{may}}$ by fact 4.2. If A is not terminated, $E[A]$ is not in **success**, therefore there is a B' such that $E[A] \mapsto B'$ and $B' \downarrow_{\text{may}}^\kappa$ for some $\kappa < \lambda$. Apply lemma **choice of execution point** to $E[A] \mapsto B'$ and show, by examination of cases (i) – (iii) , either $E[A \text{ or } A'] \mapsto B'$ and conclude by fact 4.2; or else $B' = E'[B]$ and $E[A \text{ or } A'] \mapsto E'[B \text{ or } A']$ and conclude by I.H. \square

Remark 6.7 The above proof simplifies if \lesssim_{may} is defined by quantification only over evaluation contexts without occurrences of the **or** combinator (see discussion in Section 6.3): Items (ii), $(***)$, and (d) above become superfluous. \square

The proof technique employed in the proofs of the evaluation lemmas and the above proposition is very general and powerful. Currently, everything we can prove about actions can be proved using this technique. But the proofs are lengthy and tedious and, moreover, they follow a common pattern. The simulation proof methods in Section 7 will distill this pattern and are more convenient to work with.

6.2 Basic context lemma

We shall now prove that the experimental preorders coincide with the corresponding closed contextual preorders. Thereby all the results we can show for the experimental orders, including the evaluation lemmas above, apply directly to the contextual orders also.

As explained in Section 5, lemma 5.1 establishes half of the claim. It states that finite contexts suffices for contextual testing, thus it remains to be shown that finite contexts and evaluation contexts are equally discriminative. Analogously to the proof of lemma 5.1, we show that the experimental preorders are preserved by all finite contexts. The coincidence of \sqsubseteq_m and \lesssim_m is an easy consequence by reference to lemma 5.1.

First of all, all action constructs other than **unfolding** are monotone:

Lemma 6.8 *All action constructs other than unfolding are monotone in \lesssim_m .*

Proof: Let A_1, A_2 be closed actions and assume $A_1 \lesssim_m A_2$. For all action constructs $O \neq \text{unfolding}$, let n be its arity and, for $i = 1, \dots, n$, let $F_i = O(A_1 \dots A_{i-1} [] A_{i+1} \dots A_n)$, and show $F_i[A_1] \lesssim_m F_i[A_2]$.

case $F_1 = \text{enact}$ abstraction of $[]$. (In the basic subset of AN that we consider, abstractions can only occur as immediate arguments to `enact`.) By lemma 6.4, $F_1[A_i] = \text{enact}$ abstraction of $A_i \approx A_i$. Conclude from

$$F_i[A_1] \approx A_1 \lesssim_m A_2 \approx F_i[A_2] .$$

case $F_1 = []$ and then A . For any evaluation context E , $E[A_i \text{ and then } A] = F[A_i]$, for $i = 1, 2$, where $F = E[F_1]$ is an evaluation context.

By definition of $A_1 \lesssim_m A_2$, we get $F[A_1] \ll_m F[A_2]$, i.e.

$$E[A_1 \text{ and then } A] \ll_m E[A_2 \text{ and then } A] .$$

Conclude $(A_1 \text{ and then } A) \lesssim_m (A_2 \text{ and then } A)$.

case $F_2 = A$ and then $[]$. First examine the case when A is terminated. Evaluation rule (4) gives the only out-going transition

$$A \text{ and then } A_i \mapsto \begin{cases} A & \text{if } A : \text{escape} \mid \text{fail} , \\ A_i & \text{if } A = \text{complete} . \end{cases}$$

By lemma 6.4,

$$A \text{ and then } A_i \approx \begin{cases} A & \text{if } A : \text{escape} \mid \text{fail} , \\ A_i & \text{if } A = \text{complete} . \end{cases}$$

Hence

$$\begin{aligned} (A \text{ and then } A_1) &\approx (A \text{ and then } A_2) && \text{if } A : \text{escape} \mid \text{fail} , \\ (A \text{ and then } A_1) &\lesssim_m (A \text{ and then } A_2) && \text{if } A = \text{complete} , \end{aligned}$$

by transitivity of \approx and \lesssim_m , using $A_1 \lesssim_m A_2$ in the latter case.

For arbitrary closed A , we prove $\forall \lambda < \omega. P(\lambda)$ where

$$P(\lambda) \Leftrightarrow \forall \text{closed } A. (A \text{ and then } A_1) \lesssim_m^\lambda (A \text{ and then } A_2) ,$$

by induction on λ . Let $\lambda < \omega, \forall \kappa < \lambda. P(\kappa)$ and show $P(\lambda)$:

For terminated A , the conclusion follows by the above argument. So assume A is not terminated, $E[A \text{ and then } A_1] \downarrow_m^\lambda$, and show $E[A \text{ and then } A_2] \downarrow_m$.

$m = \text{may}$: Clearly not $E[A \text{ and then } A_1] : \text{success}$. Thus $E[A \text{ and then } A_1] \mapsto B'_1$ where $B'_1 \downarrow_{\text{may}}^\kappa$ for some $\kappa < \lambda$. By lemma **choice of execution point**:

Either (i) $B'_1 = E[B_1]$ where $(A \text{ and then } A_1) \mapsto B_1$ which must be because $A \mapsto A'$ and $B_1 = (A' \text{ and then } A_1)$ since A is not terminated. Hence also $E[A \text{ and then } A_2] \mapsto E[A' \text{ and then } A_2]$. By I.H. $B'_1 \downarrow_{\text{may}}^\kappa$ implies $E[A' \text{ and then } A_2] \downarrow_{\text{may}}$ and by fact 4.2, $E[A \text{ and then } A_2] \downarrow_{\text{may}}$.

Or (ii) $B'_1 = E'[A \text{ and then } A_1]$ and $E[A \text{ and then } A_2] \mapsto E'[A \text{ and then } A_2]$.
By I.H. and fact 4.2, $E[A \text{ and then } A_2] \downarrow_{\text{may}}$.

Or (iii) $E[A \text{ and then } A_2] \mapsto B'_1$ and by fact 4.2, $E[A \text{ and then } A_2] \downarrow_{\text{may}}$.

$m = \text{must}$: Clearly $E[A \text{ and then } A_2] \neq \text{fail}$. We must furthermore show whenever $E[A \text{ and then } A_2] \mapsto B'_2$, $B'_2 \downarrow_{\text{must}}$. By lemma **choice of execution point**:

Either (i) $B'_2 = E[B_2]$ and $(A \text{ and then } A_2) \mapsto B_2$ which must be because $A \mapsto A'$ and $B_2 = (A' \text{ and then } A_2)$ since A is not terminated. Hence also $E[A \text{ and then } A_1] \mapsto E[A' \text{ and then } A_1]$. Since $E[A \text{ and then } A_1] \downarrow_{\text{must}}^\lambda$, $E[A' \text{ and then } A_2] \downarrow_{\text{must}}^\kappa$ for some $\kappa < \lambda$ and by I.H. $B'_2 \downarrow_{\text{must}}$.

Or (ii) $B'_2 = E'[A \text{ and then } A_2]$ and $E[A \text{ and then } A_1] \mapsto E'[A \text{ and then } A_1]$.
 $E'[A \text{ and then } A_1] \downarrow_{\text{must}}^\kappa$ for some $\kappa < \lambda$, hence $B'_2 \downarrow_{\text{must}}$ by I.H.

Or (iii) $E[A \text{ and then } A_1] \mapsto B'_2$ and by fact 4.2, $B'_2 \downarrow_{\text{must}}$.

This concludes case $F_2 = A \text{ and then } []$.

The $A \text{ trap } []$ case is similar to the $A \text{ and then } []$ case. The remaining cases $[] \text{ and } A$, $A \text{ and } []$, $[] \text{ or } A$, $A \text{ or } []$, $[] \text{ trap } A$ are similar to the $[] \text{ and then } A$ case (these are evaluation contexts). \square

Consequently, all finite contexts are monotone:

Proposition 6.9 *All closed, finite contexts are monotone in \lesssim_m .*

Proof: Assume $A_1 \lesssim_m A_2$ and show $F[A_1] \lesssim_m F[A_2]$, for all closed, finite F , by structural induction on F .

If $F = []$ or F has no holes, the result is immediate.

Otherwise, if $F = O(F_1 \dots F_n)$, by I.H. $F_i[A_1] \lesssim_m F_i[A_2]$, for $i = 1, 2$. O is monotone since $O \neq \text{unfolding}$, hence

$$\begin{aligned}
F[A_1] &= O(F_1[A_1] F_2[A_1] \dots F_{n-1}[A_1] F_n[A_1]) \\
&\lesssim_m O(F_1[A_2] F_2[A_1] \dots F_{n-1}[A_1] F_n[A_1]) \\
&\quad \vdots \\
&\lesssim_m O(F_1[A_2] F_2[A_2] \dots F_{n-1}[A_2] F_n[A_1]) \\
&\lesssim_m O(F_1[A_2] F_2[A_2] \dots F_{n-1}[A_2] F_n[A_2]) = F[A_2].
\end{aligned}$$

\square

We can now prove the following context lemma by the same proof technique as exercised for lemma 5.1 in Section 5.2.

Lemma 6.10 (Basic context lemma) *For closed actions A_1, A_2 ,*

$$\begin{aligned} A_1 \sqsubseteq_{\text{may}} A_2 & \text{ if and only if } A_1 \lesssim_{\text{may}} A_2 . \\ A_1 \sqsubseteq_{\text{must}} A_2 & \text{ if and only if } A_1 \lesssim_{\text{must}} A_2 . \end{aligned}$$

Proof: By reference to lemma 5.1 it suffices to show, for $m \in \{\text{may}, \text{must}\}$,

$$\forall \text{ closed, finite } F. F[A_1] \ll_m F[A_2] \Leftrightarrow A_1 \lesssim_m A_2 .$$

‘ \Rightarrow ’ is immediate because evaluation contexts are instances of finite contexts.

For ‘ \Leftarrow ’, assume $A_1 \lesssim_m A_2$ and let C be any closed, finite context. By proposition 6.9, $C[A_1] \lesssim_m C[A_2]$, and the result follows because \lesssim_m by its definition is included in \ll_m . \square

6.3 Stronger characterisations

The experimental preorders restrict the set of observing contexts to evaluation contexts. Section 6.1 demonstrates how this restriction admits proofs by induction on the length of computation (the pattern of practically all experimental order proofs).

Further restrictions of the set of observing contexts—that still characterise the contextual preorders and still generalise to all facets—may expose general properties of actions but appear to be useful for special cases only. Essential simplifications of the experimental order characterisation cannot be obtained along this path. Nevertheless, let us mention a few possible restrictions:

In the **may** case (which normally has the simplest characterisation, cf. [13]) one can show that evaluation contexts without occurrences of the **or** combinator suffice. For an example where this stronger characterisation would be convenient, see remark 6.7.

As long as we have no side-effects allowing interference between interleaved actions—i.e., when we restrict ourselves to the basic, functional, and declarative facets—contexts with no occurrences of the interleaving combinator **and** suffice.

For basic actions, things are very simple and, in fact, only a small finite number of contexts are needed to distinguish the equivalence classes of the **may** and **must** relations. The 2 **may** contexts and the 4 **must** contexts listed in Section 4.3 are both sufficient and minimal. This can be verified by examining the pictures in Section 4.3. As soon as other facets are added, infinite collections of contexts are required to characterise the contextual preorders.

7 Simulation

The proofs for showing actions experimental preordered all follow the same pattern: Construct an invariant or relation and show the actions to evaluate to related actions or to the same action. By induction on \downarrow_{may} or \downarrow_{must} the result follows.

This is a straightforward but lengthy and tedious technique. Moreover the connection to co-inductive bisimulation proof techniques is striking. In this section, simple simulation proof principles will be established for actions.

7.1 may and must simulation

The \simeq_m preorders (definition 4.6) are simple ‘simulation-style’ preorders, defined in terms of entire computations.

Because of the possibly interleaved performance of actions, their semantics is defined in terms of atomic computation steps and simulation is also most appropriately phrased in terms of individual computation steps.

We define simulation co-inductively, inspired by the bisimulation for actions in [22, C.4]. Our **may** and **must** formulation is taken from [34].

Define two operators $\langle - \rangle_{\text{may}}$ and $\langle - \rangle_{\text{must}}$:

Definition 7.1 *Given a relation $\mathcal{R} \subseteq \text{closed-action} \times \text{closed-action}$, define relations $\langle \mathcal{R} \rangle_{\text{may}}, \langle \mathcal{R} \rangle_{\text{must}} \subseteq \text{closed-action} \times \text{closed-action}$ as follows*

$$A_1 \langle \mathcal{R} \rangle_{\text{may}} A_2 \stackrel{\text{def}}{\iff} \begin{array}{l} \text{(I)} \quad A_1:\text{success} \Rightarrow A_2 \mapsto^* A_1 \\ \wedge \quad \text{(II)} \quad (\forall B_1 \mid A_1 \mapsto B_1)(\exists B_2 \mid A_2 \mapsto^* B_2) B_1 \mathcal{R} B_2 . \end{array}$$

$$A_1 \langle \mathcal{R} \rangle_{\text{must}} A_2 \stackrel{\text{def}}{\iff} \begin{array}{l} A_1 \Downarrow \Rightarrow \\ \text{(I)} \quad A_2 \Downarrow \\ \wedge \quad \text{(II)} \quad A_2:\text{terminated} \Rightarrow A_1 \mapsto^* A_2 \\ \wedge \quad \text{(III)} \quad (\forall B_2 \mid A_2 \mapsto B_2)(\exists B_1 \mid A_1 \mapsto^* B_1) B_1 \mathcal{R} B_2 . \end{array}$$

Both $\langle - \rangle_{\text{may}}$ and $\langle - \rangle_{\text{must}}$ are easily seen to be monotone.

Call a post-fixed point of $\langle - \rangle_{\text{may}}$ a ‘**may** simulation’, and a post-fixed point of $\langle - \rangle_{\text{must}}$ a ‘**must** simulation’.

Fact 7.2 *For $m \in \{\text{may}, \text{must}\}$, $\text{Id} \stackrel{\text{def}}{=} \{(A, A) \mid A:\text{closed-action}\}$ is a m simulation, and if \mathcal{R} and \mathcal{S} are m simulations, so is their composition $\mathcal{R}\mathcal{S}$.*

Definition 7.3 $\lesssim_{\text{may}} \stackrel{\text{def}}{=} \bigcup \{ \mathcal{R} \mid \mathcal{R} \text{ is a may simulation} \}$,
 $\lesssim_{\text{must}} \stackrel{\text{def}}{=} \bigcup \{ \mathcal{R} \mid \mathcal{R} \text{ is a must simulation} \}$.

Thus $A_1 \lesssim_m A_2$ if and only if there is a m simulation \mathcal{R} such that $(A_1, A_2) \in \mathcal{R}$. Call A_1 ‘ m similar to’ A_2 if $A_1 \lesssim_m A_2$. Fact 7.2 implies

Fact 7.4 \lesssim_{may} and \lesssim_{must} are reflexive and transitive, i.e. preorders.

Let ‘simulation equivalence’ denote the intersection of the induced equivalences of the two preorders. Notice that two actions are simulation equivalent if there is a symmetric relation containing them which is both a **may** and a **must** simulation; call such a relation an ‘equivalence simulation’.

Proposition 7.5 \lesssim_{may} and \lesssim_{must} are the largest **may** and **must** simulations, and the largest fixed points of $\langle - \rangle_{\text{may}}$ and $\langle - \rangle_{\text{must}}$.

Proof: See lemma 1 and theorem 1 in [10]. \square

Remark 7.6 One can also define a bisimulation preorder as in [35]: A_1 is *pre-bisimilar* to A_2 iff $(A_1, A_2) \in \nu\mathcal{R}.\langle \mathcal{R} \rangle_{\text{may}} \cap \langle \mathcal{R} \rangle_{\text{must}}$. The induced equivalence is sensitive to divergence and thus stronger than standard weak bisimulation in process calculus [20] and as defined for actions by Mosses [22, C.4]. This makes this pre-bisimulation equivalence stronger than both **may** and **must** contextual test equivalence which is not the case for standard weak bisimulation (see [20]).

The simulation preorders correspond to Ulidowski’s ‘copy+refusal testing’. For processes, Ulidowski shows that this refines our contextual preorders (but is coarser than pre-bisimulation). \square

Because the **may** and **must** simulations to some extent ‘mirror’ each other, they naturally combine into **impl** simulations. These prove very convenient and are used extensively for the development of the equational theory of actions in Section 8.1. As usual, define $\lesssim_{\text{impl}} = \gtrsim_{\text{may}} \cap \lesssim_{\text{must}}$, that is, $\lesssim_{\text{impl}} = \bigcup \{ \mathcal{R} \mid \mathcal{R} \text{ is an } \mathbf{impl} \text{ simulation} \}$ where \mathcal{R} is an **impl** simulation if \mathcal{R} is a **must** simulation and \mathcal{R}^{-1} is a **may** simulation,

$$\mathcal{R} \subseteq \langle \mathcal{R} \rangle_{\text{must}} \cap \langle \mathcal{R}^{-1} \rangle_{\text{may}}^{-1} .$$

Define an auxiliary operator $\langle - \rangle_{\text{impl}} = \langle - \rangle_{\text{must}} \cap \langle -^{-1} \rangle_{\text{may}}^{-1}$. Notice that every symmetric **impl** simulation is an equivalence simulation.

Fact 7.7 The definition of $\langle - \rangle_{\text{impl}}$ expands to

$$\begin{aligned} A_1 \langle \mathcal{R} \rangle_{\text{impl}} A_2 &\Leftrightarrow \quad \text{(I)} \quad A_1 \Downarrow \Rightarrow \quad \begin{array}{l} \text{(i)} \quad A_2 \Downarrow \\ \wedge \quad \text{(ii)} \quad A_2 = \text{fail} \Rightarrow A_1 \mapsto^* \text{fail} \end{array} \\ &\wedge \quad \text{(II)} \quad A_2 : \text{success} \Rightarrow A_1 \mapsto^* A_2 \\ &\wedge \quad \text{(III)} \quad (\forall B_2 \mid A_2 \mapsto B_2) (\exists B_1 \mid A_1 \mapsto^* B_1) B_1 \mathcal{R} B_2 . \end{aligned}$$

7.2 Simulation proof techniques

We shall now demonstrate the power of simulations by conducting a number of proofs of simulation orderings, including simulation order versions of the experimental order results from Section 6.1. The simpler simulation proofs demonstrate how **may** and **must** simulations expose the inductive backbone of the corresponding experimental order proofs.

It proves convenient to enhance the simulation proof technique in two ways:

- **must** and **impl** simulation proofs require arguments about convergence. But we shall escape the development of machinery for reasoning about convergence by defining a ‘hybrid weak/strong’ **impl** operator; it ensures that similar actions, say A_1, A_2 , simulate each other so closely that whenever A_2 diverges, so does A_1 .
- General proof techniques for greatest fixed points of monotone operators will allow us to conduct simulation proofs using improper “simulations” that are smaller than proper post-fixed points of the $\langle - \rangle_m$ operators.

This subsection concludes by applying these techniques for proving simulation versions of the evaluation lemmas (6.2) and (6.4).

Simple examples

First three examples that demonstrate the concise co-inductive simulation proof method of exhibiting simulations:

Proposition 7.8 $\text{fail} \lesssim_{\text{may}} A$, for all closed A .

Proof: $\mathcal{R} = \{(\text{fail}, A) \mid A:\text{closed-action}\}$ is a **may** simulation, i.e., for all A , $\text{fail} \langle \mathcal{R} \rangle_{\text{may}} A$, because (I) **fail** is not in **success**; and (II) there are no transitions out of **fail**. \square

The proof is much simpler than that of proposition 6.6 because $\langle - \rangle_{\text{may}}$ has been defined to make **fail** the bottom element.

diverge is the **must** bottom element because **diverge** \uparrow :

Proposition 7.9 $A_1 \uparrow$ implies $A_1 \lesssim_{\text{must}} A_2$, for all closed A_1, A_2 .

Proof: $\{(A_1, A_2) \mid A_1, A_2:\text{closed-action}, A_1 \uparrow\}$ is clearly a **must** simulation because $A_1 \uparrow$ implies $A_1 \langle \mathcal{R} \rangle_{\text{must}} A_2$, for all \mathcal{R}, A_1, A_2 . \square

chaos is both \top_{may} and \perp_{must} (cf. Section 4.3):

Proposition 7.10 $\text{chaos} \lesssim_{\text{impl}} A$, for all closed A .

Proof: $\mathcal{R} = \{(\text{chaos}, A) \mid A:\text{closed-action}\}$ is an **impl** simulation, i.e., for all A , $\text{chaos} \langle \mathcal{R} \rangle_{\text{impl}} A$, because (I) **chaos** \uparrow ; (II) **chaos** $\mapsto S$, for all $S:\text{success}$; (III) whenever $A \mapsto B_2$, let $B_1 = \text{chaos}$. \square

A ‘hybrid’ impl operator

The $\langle - \rangle_{\text{must}}$ and $\langle - \rangle_{\text{impl}}$ operators involve the convergence predicate, \Downarrow , in order to make simulations sensitive to divergence. This obligation to prove actions convergent complicates reasoning. Another solution is to resort to strong (bi)simulation which is also sensitive to divergence but requires that (bi)similar programs make exactly the same number of computation steps.

A hybrid weak/strong **impl** operator, $\langle - \rangle_{\text{impl}}^{\text{hybr}}$, is a convenient compromise:

Definition 7.11 *For every relation \mathcal{R} and closed actions A_1, A_2 ,*

$$A_1 \langle \mathcal{R} \rangle_{\text{impl}}^{\text{hybr}} A_2 \stackrel{\text{def}}{\iff} \begin{array}{l} \text{(I)} \quad A_2 \text{:terminated} \Rightarrow A_1 \mapsto^* A_2 \\ \wedge \quad \text{(II)} \quad (\forall B_2 \mid A_2 \mapsto B_2)(\exists B_1 \mid A_1 \mapsto^+ B_1) B_1 \mathcal{R} B_2 . \end{array}$$

$\langle - \rangle_{\text{impl}}^{\text{hybr}}$ is monotone. Intuitively, A_1 may perform no less steps than A_2 . Whenever $A_1 \mathcal{R} A_2$ for some $\mathcal{R} \subseteq \langle \mathcal{R} \rangle_{\text{impl}}^{\text{hybr}}$, if A_2 may diverge, so may A_1 .

The following lemma states that, in the course of proving a relation \mathcal{R} an **impl** simulation, i.e. $A_1 \mathcal{R} A_2$ implies $A_1 \langle \mathcal{R} \rangle_{\text{impl}} A_2$, it suffices to show $A_1 \langle \mathcal{R} \rangle_{\text{impl}}^{\text{hybr}} A_2$.

Lemma 7.12 $\mathcal{R} \subseteq \langle \mathcal{R} \rangle_{\text{impl}} \cup \langle \mathcal{R} \rangle_{\text{impl}}^{\text{hybr}}$ implies $\mathcal{R} \subseteq \langle \mathcal{R} \rangle_{\text{impl}}$.

Proof (sketch): It suffices to show that the antecedent implies $\langle \mathcal{R} \rangle_{\text{impl}}^{\text{hybr}} \subseteq \langle \mathcal{R} \rangle_{\text{impl}}$. This is the case if $A_1 \langle \mathcal{R} \rangle_{\text{impl}}^{\text{hybr}} A_2$ implies $A_1 \Downarrow \Rightarrow A_2 \Downarrow$. The latter follows from $\forall \mu < \omega. A_1 \Downarrow^\mu \Rightarrow A_2 \Downarrow$ and is shown by induction on μ . \square

In particular, every post-fixed point of $\langle - \rangle_{\text{impl}}^{\text{hybr}}$ is an **impl** simulation. E.g., $Id \subseteq \langle - \rangle_{\text{impl}}^{\text{hybr}}$ shows not only that the greatest fixed point of $\langle - \rangle_{\text{impl}}^{\text{hybr}}$ is reflexive but also that \lesssim_{impl} is reflexive.

Lemma 7.12 allows **impl** simulation proofs to eschew direct reasoning about convergence by establishing both the necessary convergence properties and similarity in one co-inductive argument. This idea resembles Gordon’s ‘refined’ treatment of divergence for bisimulation of functional programs [11].

Properties of greatest fixed points

Paulson [32] and Gordon [11] use the following general properties of greatest fixed points:

Proposition 7.13 *Let ν be the greatest fixed point of a monotone operator $\langle - \rangle$,*

$$\begin{aligned} \nu &\stackrel{\text{def}}{=} \nu \mathcal{R} . \langle \mathcal{R} \rangle . \text{ Then } \nu = \nu \mathcal{R} . \langle \mathcal{R} \rangle \cup \nu \\ &= \nu \mathcal{R} . \langle \mathcal{R} \cup \nu \rangle \\ &= \nu \mathcal{R} . \langle \mathcal{R} \cup \nu \rangle \cup \nu . \end{aligned}$$

A useful corollary applies to all the $\langle - \rangle$ operators we consider:

Corollary 7.14 *Whenever $\langle _ \rangle$ is monotone and $Id \subseteq \nu\mathcal{R}.\langle \mathcal{R} \rangle$,*

$$\mathcal{R} \subseteq \langle \mathcal{R} \cup Id \rangle \text{ implies } \mathcal{R} \subseteq \nu\mathcal{R}.\langle \mathcal{R} \rangle .$$

Proof: $\mathcal{R} \subseteq \langle \mathcal{R} \cup Id \rangle \Rightarrow \mathcal{R} \subseteq \langle \mathcal{R} \cup \nu\mathcal{R}.\langle \mathcal{R} \rangle \rangle$, $\langle _ \rangle$ is monotone
 $\Rightarrow \mathcal{R} \subseteq \nu\mathcal{R}.\langle \mathcal{R} \cup \nu\mathcal{R}.\langle \mathcal{R} \rangle \rangle$, by co-induction
 $\Rightarrow \mathcal{R} \subseteq \nu\mathcal{R}.\langle \mathcal{R} \rangle$, by proposition 7.13. \square

Evaluation lemmas

We now prove simulation versions of the evaluation lemmas. The proofs apply hybrid **impl** simulation and corollary 7.14.

The analogue of lemma **evaluation** (6.2) is simple to show:

Lemma 7.15 $A_1 \mapsto A_2$ *implies* $A_1 \lesssim_{\text{impl}} A_2$.

Proof: Let $\mathcal{R} = \{(A_1, A_2) \mid A_1 \mapsto A_2\}$. Use lemma 7.12 and corollary 7.14 and show $\mathcal{R} \subseteq \langle \mathcal{R} \cup Id \rangle_{\text{impl}}^{\text{hybr}}$, i.e. $A_1 \mapsto A_2$ implies $A_1 \langle \mathcal{R} \cup Id \rangle_{\text{impl}}^{\text{hybr}} A_2$: (I) A_1 is not terminated; and (II) whenever $A_2 \mapsto B_2$, also $A_1 \mapsto^2 B_1$, so choose $B_2 = B_1$. \square

The determined variant of lemma 7.15 is an example where $\langle _ \rangle_{\text{impl}}^{\text{hybr}}$ is not applicable:

Lemma 7.16 *If $A_1 \mapsto A_2$ is the only transition out of A_1 , then A_1 is simulation equivalent to A_2 .*

Proof: Given lemma 7.15 and corollary 7.14, it suffices to show $\mathcal{R} \subseteq \langle \mathcal{S} \cup Id \rangle_{\text{impl}}$ where $\mathcal{R} = \{(A_2, A_1) \mid A_1 \mapsto A_2 \text{ is the only transition out of } A_1\}$. Whenever $A_1 \mapsto A_2$ is the only transition out of A_1 , show $A_2 \langle \mathcal{R} \cup Id \rangle_{\text{impl}} A_1$, using fact 7.7: (I)(i) follows by definition 3.2 since $A_1 \mapsto A_2$ is the only transition out of A_1 . (I)(ii) and (II) hold because A_1 cannot be terminated when $A_1 \mapsto A_2$. To show (III), notice that $A_1 \mapsto B_2$ implies $B_2 = A_2$, so let $A_2 \mapsto^0 B_1 = B_2$. \square

7.3 Simulation versus contextual testing

Our interest in simulations is as a tool for proving actions contextual preordered. Therefore we have to show that the simulation preorders are included in the contextual preorders. By the **basic context lemma** (6.10) it suffices to prove that they are included in the experimental preorders. A more standard path, due to Howe [15], is to show the simulation preorders to be precongruences—as in the proof of lemma 5.1—and consequently included in the contextual preorders. Given the **basic context lemma**, our approach is considerably simpler. (Mason, Smith, and Talcott follow a similar route in [18].)

Lemma 7.17 For closed actions A_1, A_2 ,

$$\begin{array}{lcl} A_1 \lesssim_{\text{may}} A_2 & \text{implies} & A_1 \lesssim_{\text{may}}^{\lambda} A_2 . \\ A_1 \lesssim_{\text{must}} A_2 & \text{implies} & A_1 \lesssim_{\text{must}}^{\lambda} A_2 . \end{array}$$

Proof:

$m = \text{may}$: Show $\forall \lambda < \omega. \lesssim_{\text{may}} \subseteq \lesssim_{\text{may}}^{\lambda}$ by induction on λ . Let $\lambda < \omega$, assume $\forall \kappa < \lambda. \lesssim_{\text{may}} \subseteq \lesssim_{\text{may}}^{\kappa}$, and show $\lesssim_{\text{may}} \subseteq \lesssim_{\text{may}}^{\lambda}$:

Assume $A_1 \lesssim_{\text{may}} A_2$, hence $A_1 \langle \lesssim_{\text{may}} \rangle_{\text{may}} A_2$ because \lesssim_{may} is a **may** simulation. If A_1 is terminated, either $A_1 = \text{fail}$ in which case the result is immediate because **fail** is the **may** bottom element (proposition 6.6); or else $A_1 \cdot \text{success}$ and $A_2 \mapsto^* A_1$, then conclude by lemma **evaluation** (6.2). If A_1 is not terminated, assume $E[A_1] \downarrow_{\text{may}}^{\lambda}$ and show $E[A_2] \downarrow_{\text{may}}$. $E[A_1]$ is not in success since A_1 is not terminated, therefore $E[A_1] \mapsto B'$ and $B' \downarrow_{\text{may}}^{\kappa}$ for some $\kappa < \lambda$. By lemma **choice of execution point**:

Either (i) $B' = E[B_1]$ where $A_1 \mapsto B_1$ and $A_2 \mapsto^* B_2$ such that $B_1 \lesssim_{\text{may}} B_2$. By I.H. $E[B_1] \downarrow_{\text{may}}^{\kappa} \Rightarrow E[B_2] \downarrow_{\text{may}}$. Conclude by fact 4.2.

Or (ii) $B' = E'[A_1]$ and $E[A_2] \mapsto E'[A_2]$. The result follows by I.H. and fact 4.2.

Or (iii) $E[A_2] \mapsto B'$. Conclude by fact 4.2.

$m = \text{must}$: The **must** case is more complicated. If we try to prove $A_1 \lesssim_{\text{must}} A_2$ implies $A_1 \lesssim_{\text{must}}^{\lambda} A_2$ by induction on λ , as in the **may** case, we run into a problem: When $A_2 \mapsto B_2$ we are not guaranteed that $A_1 \mapsto^+ B_1$ such that $B_1 \lesssim_{\text{must}} B_2$. We may only have that $A_1 \lesssim_{\text{must}} B_2$ and then we cannot refer to the induction hypothesis to perform the induction step.

Therefore construct \lesssim_{must} as the limit of an increasing sequence,

$$\begin{aligned} \lesssim_{\text{must}} &= \bigcup_{\mu < \omega} \lesssim_{\text{must}}^{\mu} \\ \text{where } A_1 \lesssim_{\text{must}}^{\mu} A_2 &\Leftrightarrow A_1 \lesssim_{\text{must}} A_2 \wedge (A_1 \downarrow \Rightarrow A_2 \downarrow^{\mu}) . \end{aligned}$$

Show $\forall \lambda, \mu < \omega. \lesssim_{\text{must}}^{\mu} \subseteq \lesssim_{\text{must}}^{\lambda}$, by well-founded induction on (λ, μ) , ordered lexicographically,

$$(\lambda_1, \mu_1) \prec (\lambda_2, \mu_2) \Leftrightarrow \lambda_1 < \lambda_2 \vee (\lambda_1 = \lambda_2 \wedge \mu_1 < \mu_2) .$$

Let $\lambda, \mu < \omega$, assume $\forall (\kappa, \nu) \prec (\lambda, \mu). \lesssim_{\text{must}}^{\nu} \subseteq \lesssim_{\text{must}}^{\kappa}$, and show $\lesssim_{\text{must}}^{\mu} \subseteq \lesssim_{\text{must}}^{\lambda}$: Assume $A_1 \lesssim_{\text{must}}^{\mu} A_2$. If not $A_1 \downarrow$, neither $E[A_1] \downarrow_{\text{must}}$, hence $A_1 \lesssim_{\text{must}} A_2$. Henceforth assume $A_1 \downarrow$. Since \lesssim_{must} is a fixed point for $\langle - \rangle_{\text{must}}$, $A_1 \lesssim_{\text{must}}^{\mu} A_2$ implies

$$\begin{aligned} & \text{(I) } A_2 \downarrow^{\mu} \\ \wedge & \text{(II) } A_2 \cdot \text{terminated} \Rightarrow A_1 \mapsto^* A_2 \\ \wedge & \text{(III) } (\forall B_2 \mid A_2 \mapsto B_2)(\exists B_1 \mid A_1 \mapsto^* B_1) B_1 \lesssim_{\text{must}} B_2 . \end{aligned}$$

If A_2 is terminated, conclude by (II) and lemma **evaluation** (6.2). If A_2 is not terminated, assume $E[A_1] \downarrow_{\text{must}}^\lambda$ and show $E[A_2] \downarrow_{\text{must}}$. Clearly $E[A_2] \neq \text{fail}$ since A_2 is not terminated. Show $E[A_2] \mapsto B'$ implies $B' \downarrow_{\text{must}}$ by lemma **choice of execution point**:

Either (i) $B' = E[B_2]$ where $A_2 \mapsto B_2$ and, according to (III), $A_1 \mapsto^* B_1$ such that $B_1 \lesssim_{\text{must}} B_2$. Either $A_1 = B_1$ and $E[B_1] \downarrow_{\text{must}}^\lambda$, then use that (I) and $A_2 \mapsto B_2$ imply $B_2 \downarrow^\nu$ for some $\nu < \mu$, thus $B_1 \lesssim_{\text{must}}^\nu B_2$. Or else $A_1 \mapsto^+ B_1$ and $E[B_1] \downarrow_{\text{must}}^\kappa$, for some $\kappa < \lambda$. In any case, we can apply I.H. to get $E[B_2] \downarrow_{\text{must}}$ as required.

Or (ii) $B' = E[A_2]$ and $E[A_1] \mapsto E'[A_1]$, hence $E'[A_1] \downarrow_{\text{must}}^\kappa$, for some $\kappa < \lambda$. Conclude by I.H.

Or (iii) $E[A_1] \mapsto B'$. Conclude by fact 4.2. □

7.4 Identity of all characterisations

So far we have shown, for $m \in \{\text{may}, \text{must}\}$, \lesssim_m is included in \approx_m (lemma 7.17) that coincide with \sqsubset_m (**basic context lemma**) which is included in \rightsquigarrow_m (lemma 4.8). Now we will show that for closed basic actions all four pairs of preorders are identical. The missing link in the chain of mutual inclusions is:

Lemma 7.18 *For closed actions A_1, A_2 ,*

$$\begin{aligned} A_1 \rightsquigarrow_{\text{may}} A_2 & \text{ implies } A_1 \lesssim_{\text{may}} A_2 . \\ A_1 \rightsquigarrow_{\text{must}} A_2 & \text{ implies } A_1 \lesssim_{\text{must}} A_2 . \end{aligned}$$

Proof: First show that $\rightsquigarrow_{\text{may}}$ is a **may** simulation, $\rightsquigarrow_{\text{may}} \subseteq \langle \rightsquigarrow_{\text{may}} \rangle_{\text{may}}$, i.e. $A_1 \rightsquigarrow_{\text{may}} A_2$ implies $A_1 \langle \rightsquigarrow_{\text{may}} \rangle_{\text{may}} A_2$. The latter expands to

$$\begin{aligned} & \text{(I) } A_1:\text{success} \Rightarrow A_2 \mapsto^* A_1 \\ \wedge & \text{(II) } (\forall B_1 \mid A_1 \mapsto B_1)(\exists B_2 \mid A_2 \mapsto^* B_2) B_1 \rightsquigarrow_{\text{may}} B_2 . \end{aligned}$$

(I) clearly follows from $A_1 \rightsquigarrow_{\text{may}} A_2$. Regarding (II), if $A_1 \mapsto B_1$, choose $B_2 = A_2$; then $B_1 \rightsquigarrow_{\text{may}} B_2$: Every successful computation $B_1 \mapsto^* S$ extends to a successful computation $A_1 \mapsto B_1 \mapsto^* S$. Since $A_1 \rightsquigarrow_{\text{may}} A_2$, there is also a computation $A_2 \mapsto^* S$, i.e. $B_2 \mapsto^* S$.

Correspondingly for **must**, assume $A_1 \rightsquigarrow_{\text{must}} A_2$ and show

$$\begin{aligned} A_1 \downarrow & \Rightarrow \text{(I) } A_2 \downarrow \\ & \wedge \text{(II) } A_2:\text{terminated} \Rightarrow A_1 \mapsto^* A_2 \\ & \wedge \text{(III) } (\forall B_2 \mid A_2 \mapsto B_2)(\exists B_1 \mid A_1 \mapsto^* B_1) B_1 \rightsquigarrow_{\text{must}} B_2 . \end{aligned}$$

The result is immediate if not $A_1 \downarrow$, so assume $A_1 \downarrow$ and hence also (I) since $A_1 \rightsquigarrow_{\text{must}} A_2$. The rest is as for the **may** case above: (II) clearly follows from $A_1 \rightsquigarrow_{\text{must}} A_2$. As for (III), when $A_2 \mapsto B_2$, let $B_1 = A_1$, then $B_1 \rightsquigarrow_{\text{must}} B_2$ because every successful computation from B_2 extends to a successful computation from A_2 and has a corresponding successful computation from $A_1 = B_1$. □

8 (In)equational action theory

This section shows how existing equational action theory respects contextual equivalence. Moreover, Section 8.2 sketches inequational theories for the preorders and demonstrate their expressive power. They enrich action theory by interesting characterisations of the behaviour of actions.

8.1 Equational theory

A number of the equational action laws from Section 2.4 were shown to respect experimental equivalence ‘ \approx ’ in Section 6.1. So by the **basic context lemma** (6.10), these laws also respect contextual equivalence ‘ \simeq ’. We shall now use that simulation equivalence is also included in contextual equivalence (lemma 7.17) to prove the remaining laws from Section 2.4.

Eq.law (9) $(A_1 \text{ or } A_2) \text{ or } A_3 \simeq A_1 \text{ or } (A_2 \text{ or } A_3)$.

Proof: Every instance of the law is included in $\mathcal{S} = \mathcal{R} \cup \mathcal{R}^{-1}$, where

$$\mathcal{R} = \{((A_1 \text{ or } A_2) \text{ or } A_3, A_1 \text{ or } (A_2 \text{ or } A_3)) \mid A_1, A_2, A_3:\text{closed-action}\} .$$

\mathcal{S} is symmetric and by lemma 7.12 and corollary 7.14 it suffices to show $\mathcal{S} \subseteq \langle \mathcal{S} \cup Id \rangle_{\text{impl}}^{\text{hybr}}$ which is straightforward. \square

Laws (3), (6), and (11) can be proved analogously.

Half of laws (4) and (7) were shown in corollary 6.5. Of law (4) remains:

Eq.law (4)b $A \text{ and then complete} \simeq A$.

Proof: Show that the symmetric relation $\mathcal{S} = \mathcal{R} \cup \mathcal{R}^{-1}$, where

$$\mathcal{R} = \{(A \text{ and then complete}, A) \mid A:\text{closed-action}\} ,$$

satisfies $\mathcal{S} \subseteq \langle Id \rangle_{\text{impl}} \cup \langle \mathcal{S} \rangle_{\text{impl}}^{\text{hybr}}$.

$\mathcal{R} \subseteq \langle \mathcal{S} \rangle_{\text{impl}}^{\text{hybr}}$ is easy to establish.

Consider \mathcal{R}^{-1} next. $A \langle \mathcal{S} \rangle_{\text{impl}}^{\text{hybr}} (A \text{ and then complete})$ holds whenever there is a transition out of A . Otherwise show $A \langle Id \rangle_{\text{impl}} (A \text{ and then complete})$ directly from fact 7.7:

Clearly $A \Downarrow$ and $(A \text{ and then complete}) \Downarrow$ when there are no transitions out of A . Only if A is terminated can $(A \text{ and then complete})$ perform a transition, $(A \text{ and then complete}) \mapsto A$, and then conclude since $A \mapsto^0 A$. \square

The proof of law (7)b is similar.

Laws (10) and (12) require a fact about convergence of or:

Fact 8.1 $A_1 \Downarrow \wedge A_2 \Downarrow$ iff $(A_1 \text{ or } A_2) \Downarrow$, for all closed A_1, A_2 .

This is proved by induction on \Downarrow . The ‘if’ direction is an instance of a more general fact:

Fact 8.2 $E[A] \Downarrow$ implies $A \Downarrow$, for all evaluation contexts E and closed A .

Eq.law (10) $\text{fail or } A \simeq A \text{ or fail} \simeq A$.

Proof: Corollary 6.3 and the commutativity of **or**, eq.law (11), established most of the law. What remains to be shown is $A \sqsubseteq_{\text{impl}} \text{fail or } A$. Therefore, show $\mathcal{R} \subseteq \langle \mathcal{R} \cup Id \rangle_{\text{impl}}$ for $\mathcal{R} = \{(A, \text{fail or } A) \mid A:\text{closed-action}\}$:

Since $\text{fail} \Downarrow$, $A \Downarrow$ implies $(\text{fail or } A) \Downarrow$. If $(\text{fail or } A) \mapsto B$, either $B = (\text{fail or } A')$ and $A \mapsto A'$ and $A' \mathcal{R} B$, or else $B = A$ and conclude since $A \mapsto^0 A$ and $A Id B$. \square

Eq.law (12) $A \text{ or } A \simeq A$.

Proof: Split the proof into $(A \text{ or } A) \sqsubseteq_{\text{impl}} A$ and $A \sqsubseteq_{\text{impl}} (A \text{ or } A)$.

For the former, $\mathcal{R} = \{(A \text{ or } A, A) \mid A:\text{closed-action}\}$ is easily shown to satisfy $\mathcal{R} \subseteq \langle \mathcal{R} \rangle_{\text{impl}}^{\text{hybr}}$.

For the latter, show $\mathcal{R} \subseteq \langle \mathcal{R} \cup Id \rangle_{\text{impl}}$ for $\mathcal{R} = \{(A, A_1 \text{ or } A_2) \mid A \mapsto^* A_1 \wedge A \mapsto^* A_2\}$, i.e. $A \langle \mathcal{R} \rangle_{\text{must}} (A_1 \text{ or } A_2)$ if $A \mapsto^* A_1 \wedge A \mapsto^* A_2$:

$A \Downarrow$ implies $A_i \Downarrow$, by fact 4.2, which implies $(A_1 \text{ or } A_2) \Downarrow$, by fact 8.1. If $(A_1 \text{ or } A_2) \mapsto B$, either:

- $B = (A'_1 \text{ or } A_2)$ and $A_1 \mapsto A'_1$, let $A \mapsto^0 A$, then $A \mathcal{R} B$ since $A \mapsto^* A_1 \mapsto A'_1$;
- symmetrically, $B = (A_1 \text{ or } A'_2)$ and $A_2 \mapsto A'_2$; or
- $B = A_i$, let $A \mapsto A_i$, then $A_i Id B$.

\square

An interesting fact about law (12) is that it doesn’t respect bisimulation equivalence. E.g., **terminate** is not bisimilar to $A = (\text{terminate or terminate})$ because $A \mapsto^5 \text{succceed}$ whereas **terminate** cannot evaluate to an action with the same processing capabilities.

All other equational action laws appear to be included in bisimulation equivalence (as conjectured by Mosses [22, C.4]).

8.2 Inequational theory

Many interesting inequational laws can be expressed in terms of the various semantic preorders.

- The equivalence classes and orderings depicted in Section 4.3 can be expressed as inequational laws.
- Lemma **evaluation** (6.2) gives an inequational characterisation of the evaluation of actions.
- The action combinators enjoy certain inequational algebraic properties.
- Induction rules characterise **unfolding** as the least fixed point combinator.

Such laws complement the existing action laws and further characterise the non-deterministic behaviour of actions.

Several of these laws were established in Section 7.2, e.g. **fail** is the **may** bottom element:

Ineq.law (i) $\text{fail} \sqsubseteq_{\text{may}} A$.

This was proved using the **may** experimental preorder in proposition 6.6 and using the **may** simulation preorder in proposition 7.8.

Since \sqsubseteq_{may} is a precongruence (by definition), in particular **or** is monotone in \sqsubseteq_{may} and the following law is a simple consequence:

Ineq.law (ii) $A_1, A_2 \sqsubseteq_{\text{may}} (A_1 \text{ or } A_2)$.

Proof: $A_1 \sqsubseteq_{\text{may}} (A_1 \text{ or } \text{fail})$, **fail** is unit for **or**, eq.law (10)
 $\sqsubseteq_{\text{may}} (A_1 \text{ or } A_2)$, using ineq.law (i) and **or** is monotone.

$A_2 \sqsubseteq_{\text{may}} (A_1 \text{ or } A_2)$ follows by commutativity of **or**, eq.law (11). □

(This law was actually established in the proof of proposition 6.6.)

In conjunction with the idempotency of **or** we have that **or** is the **may** least upper bound operator, ' \sqcup_{may} ': it is an upper bound by ineq.law (ii); it is the least such because $A_1, A_2 \sqsubseteq_{\text{may}} A$ implies

$$\begin{aligned} (A_1 \text{ or } A_2) &\sqsubseteq_{\text{may}} (A \text{ or } A_2) \quad , \text{ by monotonicity of } \text{or} \\ &\sqsubseteq_{\text{may}} (A \text{ or } A) \quad , \text{ by monotonicity of } \text{or} \\ &\simeq_{\text{may}} A \quad , \text{ by idempotency of } \text{or}, \text{ eq.law (12)}. \end{aligned}$$

Section 4.3 argues that the \oplus operator is \sqcup_{may} , thus \oplus coincides with the **or** combinator **may**-wise. **must**-wise, only the inequality $(A_1 \oplus A_2) \sqsubseteq_{\text{must}} (A_1 \text{ or } A_2)$ holds. The other direction fails because $(\text{complete} \oplus \text{fail}) \not\sqsubseteq_{\text{must}} \text{complete}$ but $\text{complete} \simeq (\text{complete} \text{ or } \text{fail})$, by eq.law (10). (Thus the inequational theory of \oplus and **or** is the same as that of \oplus and $+$ in [13].)

As another example of an inequational characterisation of action combinators, we can express an implementation relationship between **and**, and **then**:

Ineq.law (iii) A_1 and $A_2 \sqsubseteq_{\text{impl}} A_1$ and then A_2 .

Proof: $\mathcal{R} = \{(A_1 \text{ and } A_2, A_1 \text{ and then } A_2) \mid A_1, A_2:\text{closed-action}\}$ is easily seen to satisfy $\mathcal{R} \subseteq \langle \mathcal{R} \cup Id \rangle_{\text{impl}}^{\text{hybr}}$. \square

Finally, we show an induction rule for the recursive combinator **unfolding**. It is a least fixed point combinator in both the **may** and **must** orderings as expressed by the following recursion induction rule (a.k.a. Park induction):

Rule 8.3 (Recursion induction) For all A , closed B , and $m \in \{\text{may}, \text{must}\}$,

$$A@B \sqsubseteq_m B \Rightarrow \text{unfolding } A \sqsubseteq_m B .$$

Proof: Assume $A@B \sqsubseteq_m B$. Let $U = \text{unfolding } A$. Notice that $U \mapsto A@U$ is the only transition out of U . By lemma 5.1 **unfolding** $A \sqsubseteq_m B$ is equivalent to $\forall \lambda < \omega. P(\lambda)$, where $P(\lambda) \Leftrightarrow \forall \text{closed } F. F[U] \ll_m^\lambda F[B]$, which we show by induction on λ . Let $\lambda < \omega$, assume $\forall \kappa < \lambda. P(\kappa)$ and show $P(\lambda)$. I.e. assume $F[U] \downarrow_m \lambda$ and show $F[B] \downarrow_m$.

$m = \text{may}$: If $F[U]:\text{success}$, then $F[U] = F = F[B]$, hence $F[B] \downarrow_{\text{may}}$. Otherwise, $F[U_1] \mapsto B'$, $B' \downarrow_{\text{may}}^\kappa$ for some $\kappa < \lambda$. By lemma 3.5:

Either (i) $B' = (E@U)[A@U]$, $F = E@[]$. Observe $B' = E[A]@U$. By I.H. and fact 5.3, $B' \downarrow_{\text{may}}^\kappa$ implies $E[A]@B \downarrow_{\text{may}}$, i.e. $(E@B)[A@B] \downarrow_{\text{may}}$. By assumption $A@B \sqsubseteq_{\text{may}} B$, hence $(E@B)[B] \downarrow_{\text{may}}$. Conclude since $(E@B)[B] = F[B]$.

Or (ii) $B' = F'[U]$ and $F[B] \mapsto F'[B]$. Since $F'[U] \downarrow_{\text{may}}^\kappa$, $F'[B] \downarrow_{\text{may}}$ by I.H., hence $F[B] \downarrow_{\text{may}}$ by fact 4.2.

$m = \text{must}$: By lemma 3.4:

Either (i) There exists an evaluation context E s.t. $E@[] = F$. Hence $F[U] = (E@U)[U] \mapsto (E@U)[A@U] = E[A]@U$ and $E[A]@U \downarrow_{\text{must}}^\kappa$, for some $\kappa < \lambda$. By I.H. and fact 5.3, $E[A]@B \downarrow_{\text{must}}$. $E[A]@B = (E@B)[A@B]$ and by assumption $A@B \sqsubseteq_{\text{must}} B$, hence $(E@B)[B] \downarrow_{\text{must}}$, i.e. $F[B] \downarrow_{\text{must}}$.

(This covered the case $F = []$. Hence $F[B] = \text{fail}$ only if $F[B] = F[U] = F = \text{fail}$ in conflict with the assumption $F[U] \downarrow_{\text{must}}$. What remains to be shown is $F[B] \mapsto B'$ implies $B' \downarrow_{\text{must}}$.)

Or (ii) whenever $F[B] \mapsto B'$, there exists F' s.t. $B' = F'[B]$ and $F[U] \mapsto F'[U]$. $F'[U] \downarrow_{\text{must}}^\kappa$ for some $\kappa < \lambda$, hence by I.H. $B' \downarrow_{\text{must}}$. Conclude $F[B] \downarrow_{\text{must}}$. \square

Normally, see e.g. [13], recursion induction is derived from a stronger induction rule, either Scott induction or an ω -rule. These stronger induction principles do not hold in the presence of countably branching nondeterminism and will therefore not generalise to the functional facet of AN, whereas our direct proof of fixed point induction does carry over (in the **must** case by means of transfinite induction over all recursive ordinals, see Section 10).

9 Stuck actions

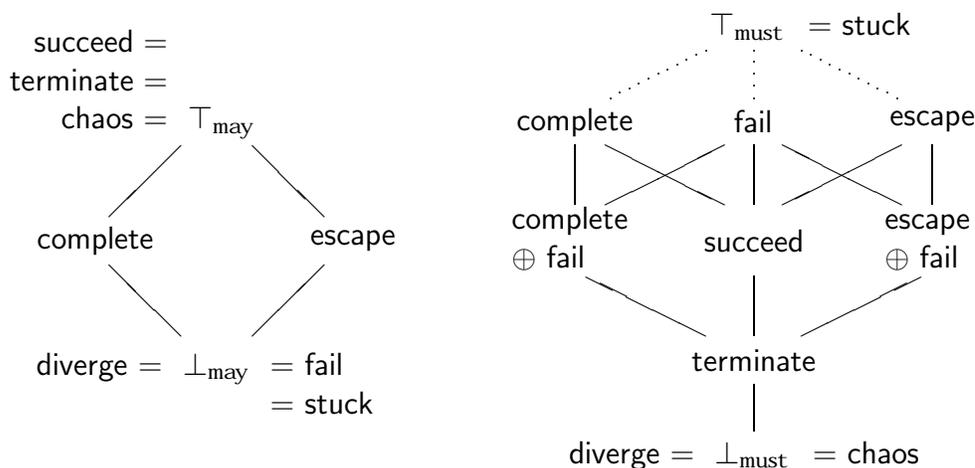
Hitherto we have assumed that closed actions are always well-behaved, that is, they never get stuck and all their closed subterms are well-behaved. In this section we discuss the problems concerned with accounting for stuck actions in our theory. Although ill-behaved actions are arguably of little interest, the problems of dealing with them shed light on the definitions and mutual consistency of the various semantic preorders as well as the definition of the evaluation relation.

Stuck theory

In AN, *failure* represents controlled error situations during execution, whereas if execution gets *stuck* it is a pathological error and can thus be viewed as uninteresting. Various syntactic requirements can be used to ensure well-behavedness.

Nonetheless, how would stuck actions fit into our theory as developed above?

Clearly all closed stuck actions are indistinguishable and thus equivalent. $\mathbf{stuck} = \mathbf{enact\ fail}$ is a representative of this equivalence class. \mathbf{stuck} can be plotted into the diagrams of the closed basic orderings as follows:



\mathbf{stuck} is the bottom element in the **may** order—intuitively, \mathbf{stuck} can never contribute ‘positively’ to produce a successful outcome and is thus equated with **diverge** and **fail**.

In the **must** order, \mathbf{stuck} is the top element, perhaps surprisingly—notice that $\mathbf{stuck} \downarrow_{\text{must}}$ and $\mathbf{stuck} \Downarrow$; examination of the proofs of proposition 4.7 and lemma 4.8 clarifies why $\mathbf{stuck} = \top_{\text{must}}$.

All the four preorders, \sqsubseteq_m , \lesssim_m , \approx_m , and \lesssim_m , still coincide when \mathbf{stuck} is taken into account.

Stuck convergence

It doesn't appear very satisfactory that **stuck** is the **must** top element. It seems more appropriate to identify **stuck** with the bottom element **diverge**—as in the **may** ordering. Can this be accomplished?

One reason why **stuck** is the top element of the four **must** preorders is that **stuck** \downarrow_{must} and **stuck** \Downarrow . The \downarrow_{must} and \Downarrow predicates can be redefined such that this is not the case, e.g. modify them to become the least predicates $\downarrow'_{\text{must}}$ and \Downarrow' satisfying

$$\begin{aligned} A \Downarrow' &\Leftrightarrow (A:\text{terminated} \vee \exists B. A \mapsto B) \wedge (\forall B | A \mapsto B) B \Downarrow' . \\ A \downarrow'_{\text{must}} &\Leftrightarrow (A:\text{success} \vee \exists B. A \mapsto B) \wedge (\forall B | A \mapsto B) B \downarrow'_{\text{must}} . \end{aligned}$$

The latter is more conveniently expressed as

$$A \downarrow'_{\text{must}} \Leftrightarrow A \downarrow_{\text{may}} \wedge (\forall B | A \mapsto B) B \downarrow'_{\text{must}} .$$

Replace \downarrow_{must} and \Downarrow by $\downarrow'_{\text{must}}$ and \Downarrow' in the definitions of the four **must** preorders and write the resulting preorders as $\sqsubseteq'_{\text{must}}$, \lesssim'_{must} , \approx'_{must} , and \sim'_{must} .

In the modified contextual and experimental preorders, $\sqsubseteq'_{\text{must}}$ and \lesssim'_{must} , **stuck** remains a maximal element above **fail** (think of the context $C_{\text{stuck}} = []$ and **escape**) but it is no longer the top element. Pictorially, remove the lines from **complete** and **escape** to **stuck** in the above **must** diagram.

In the 'simulation-style' preorders, \approx'_{must} , and \sim'_{must} , **stuck** becomes the bottom element but the preorders are no longer precongruences (recall C_{stuck}).

C_{stuck} illustrates the problem that makes **stuck** maximal in $\sqsubseteq'_{\text{must}}$ and \lesssim'_{must} and makes \approx'_{must} , and \sim'_{must} non-congruent, namely the unfortunate interplay between nondeterminism in the operational semantics and stuck redexes:

Whenever there are more redexes to choose from, a non-stuck redex is chosen. The existence of a stuck redex therefore forces evaluation to choose any alternative redex. Hence stuck subterms may control the execution of a compound action without causing the compound action to get stuck. E.g., (**stuck and escape**) is indistinguishable from **escape**.

10 Generalisations to other facets

We have been dealing with four preorders: \sqsubseteq_m , \lesssim_m , \lesssim_m , and \approx_m , for $m \in \{\text{may}, \text{must}\}$. And we have shown that they are all equivalent for closed basic actions,

$$\approx_m \supseteq^{(1)} \sqsubseteq_m =^{(2)} \lesssim_m \supseteq^{(3)} \lesssim_m \supseteq^{(4)} \approx_m ,$$

where facts (1), (2), (3), and (4) were established in lemma 4.8, **basic context lemma** 6.10, lemma 7.17, and lemma 7.18, respectively. Two proof techniques for establishing actions contextual preordered have come out of this:

- Show them experimental ordered by induction on the length of computation using lemma **choice of execution point** (3.6), conclude by the **basic context lemma**.
- Exhibit a simulation containing them, conclude by lemma 7.17.

How does all this machinery generalise to other facets of AN?

In general, \sqsubset_m , \lesssim_m , and the **basic context lemma** are straightforward to generalise whereas the other ‘simulation-style’ preorders are more language-specific.

The functional and declarative facets

When the functional or declarative facet is added, higher-order data flow is introduced.

This adds a complication to the proof of the **basic context lemma**. The proof technique from the proof of the ‘**ciu**’ theorem in [14] has to be applied.

\approx_m has to be defined co-inductively (in the style of Ong [28]’s bisimulation preorder for a nondeterministic λ -calculus). The co-inductive definition of \lesssim_m must be generalised likewise.

All the techniques, results, and laws we have developed for basic actions generalise to these facets.

The functional facet also introduces unbounded, countable nondeterminism. This is easily accommodated in our theory. The only major change is that the inductive definitions of \downarrow_{must} and $\ll_{\text{must}}^\lambda$ must be defined as transfinite limits,

$$\downarrow_{\text{must}} = \bigcup_{\lambda < \Omega} \downarrow_{\text{must}}^\lambda \quad , \quad \ll_{\text{must}} = \bigcap_{\lambda < \Omega} \ll_{\text{must}}^\lambda \quad ,$$

where Ω is the least non-recursive ordinal, cf. [2]. All results and proofs in this report have also been carefully phrased so that they carry over. Simply replace all induction arguments in the **must** case by transfinite induction over all recursive ordinals.

The imperative and communicative facets

The imperative and communicative facets introduce actions with side-effects and interacting with their context.

We have extended the **basic context lemma** to the imperative facet along the lines of [14]. As for the basic facet, nondeterminism can be handled using a generalisation of lemma **choice of execution point** (the AN concept of ‘commitment’ [22] has to be incorporated). We still have not considered the communicative facet but we believe that [1] should be readily applicable.

The bisimulation for actions in [22, C.4] indicates how \approx_m and \lesssim_m can be generalised. Then lemma 7.18 will cease to hold. Ulidowski [34] explains how

\lesssim_m (his ‘copy+refusal testing’) refines our contextual testing but is weaker than bisimulation. Thus \lesssim_m becomes strictly stronger than \sqsubseteq_m but is a tighter approximation than bisimulation.

All the action laws of Section 8 also hold for these facets. The evaluation lemmas 6.2 and 6.4 only address transitions that can be expressed locally, independent of context. Lemma **evaluation** (6.2) remains true—when suitably generalised—for transitions performed by ‘interacting’ imperative and communicative actions. Lemma **determined evaluation** (6.4) doesn’t; this lemma in effect states that, for observing contexts, execution in parallel with the transition being observed adds no discriminative power to observation. This is, in general, false for side-effects.

11 Conclusion

We conclude by recapitulating the technical exposition of this report and by relating our approach to other semantic approaches. Finally, we outline future work.

Overview

We have defined contextual test preorders for actions and developed useful operational proof techniques for establishing them.

Firstly, proofs by induction on the length of computation, based on an alternative characterisation of the preorders in terms of ‘experimental’ preorders. We have extended this proof technique, due to Mason and Talcott, to the nondeterminism in AN.

Secondly, simple simulation proofs. We use divergence sensitive simulations in order to match the contextual preorders. To escape the resulting complexity we employed a ‘hybrid’ simulation technique.

The standard range of test preorders and equivalences allowed us to exhibit an induction rule and inequational and equational laws that give rich characterisations of actions. In particular, we have shown that existing action laws hold for contextual equivalence.

Since basic actions are not very expressive, basic action theory is not very interesting in itself. But we believe that all the theory which we have presented for basic actions generalises to full AN. We have already developed this for the functional, declarative, and imperative facets—we have defined reduction semantics, proved generalisations of the **(basic) context lemma** (6.10), and proved existing action laws to hold for contextual equivalence. The experimental order proof technique used in the basic facet as well as the evaluation lemma **evaluation** (6.2) carry over to these other facets too. All the inequational and equational basic action laws in Section 8 remain true when other facets are added.

The simulation proof methods must be adjusted when imperative side-effects enter the picture and the simulation proof methods become incomplete.

Related work

Our operational techniques draw inspiration from a wide range of sources. Most of this literature is concerned with developing special theories for particular languages. Our aspiration is to obtain a useful, general action theory for reasoning about a large class of programming languages. The same objective is pursued by the domain theoretic meta language used in conventional denotational semantics and the associated reasoning techniques. The claim of action semantics is that actions have better pragmatic properties as a semantic meta language, in particular, for descriptions of realistic, complex programming languages. Motivated by these qualities of actions, our work investigates how actions can also provide a useful foundation for program reasoning.

Above we have reworked and strengthened the foundations of the existing action theory in [22] which addresses full AN, based on a nondeterministic structural operational semantics and bisimulation. Alternative semantic foundations for actions exist but none of these appear extensible to a comprehensive theory of full AN. In [7] a categorical semantics is defined for a typed subset of AN and a number of action laws are proven. For a similar subset of AN, Doh and Schmidt [6] define a versatile natural semantics framework for reasoning about (contextual) semantic equivalences (among many other things). Palsberg [30] and Moura [26] define natural semantics for larger AN subsets. Moura defines a ‘functional’ action equivalence but not a full action theory.

Future work

We plan to extend our work to full AN, as sketched in Section 10. Moreover, we are going to explore various applications of the theory. We have two sorts of applications in mind:

First of all, in order to compare action theory to conventional operational and denotational reasoning principles, we want to apply action theory to the pure, well-understood programming languages studied in traditional semantic literature.

Secondly, we would like to exploit the power of action semantics for describing large, complex languages in popular use. Hopefully, a strong action theory will provide means to apply rigorous semantic reasoning to such languages.

Acknowledgements I owe thanks to Peter Mosses for helpful suggestions that improved the presentation of this work, to Peter Ørbæk for discussions of problems encountered in the course of the work, and to Jaap van Oosten for explaining transfinite ordinals to me.

References

- [1] G. Agha, I. A. Mason, S. F. Smith, and C. L. Talcott. A foundation for actor computation, Aug. 1994. To appear in *Journal of Functional Programming*.
- [2] K. R. Apt and G. D. Plotkin. Countable nondeterminism and random assignment. *J.ACM*, 33(4):724–767, 1986.
- [3] J. Buhl. Communicative action semantics. M.Sc. dissertation, Computer Science Department, Aarhus University, Nov. 1994.
- [4] R. DeNicola and M. Hennessy. Testing equivalences for processes. *Theoretical Comput. Sci.*, 34:83–133, 1984.
- [5] K.-G. Doh and D. Schmidt. Action semantics-directed prototyping. *Comput. Lang.*, 19(4):213–233, 1993.
- [6] K.-G. Doh and D. Schmidt. The facets of action semantics: Some principles and applications. In Mosses [23], pages 1–15.
- [7] S. Even and D. A. Schmidt. Category sorted algebra-based action semantics. *Theoretical Comput. Sci.*, 77:73–96, 1990.
- [8] M. Felleisen and D. P. Friedman. Control operators, the SECD-machine, and the λ -calculus. In M. Wirsing, editor, *Formal Description of Programming Concepts III*. IFIP, 1987.
- [9] M. Felleisen and R. Hieb. The revised report on the syntactic theories of sequential control and state. *Theoretical Comput. Sci.*, 103:235–271, 1992.
- [10] A. D. Gordon. A tutorial on co-induction and functional programming. In *Glasgow Workshop on Functional Programming*, 1994.
- [11] A. D. Gordon. Bisimulation as a theory of functional programming. In *Proceedings of the 11th Conference of Mathematical Foundations of Programming Semantics*, volume 1 of *Electronic Notes in Computer Science*. Elsevier, 1995.
- [12] B. S. Hansen and J. U. Toft. The formal specification of ANDF, an application of action semantics. In Mosses [23], pages 34–42.
- [13] M. Hennessy. *Algebraic Theory of Processes*. MIT Press, 1988.
- [14] F. Honsell, I. A. Mason, S. F. Smith, and C. L. Talcott. A variable typed logic of effects, 1993. To appear in *Information and Computation*.
- [15] D. J. Howe. Equality in lazy computation systems. In *4th Annual Symposium on logic in computer science*. IEEE, 1989.

- [16] S. B. Lassen. Design and semantics of action notation. In Mosses [23], pages 16–33.
- [17] S. B. Lassen. Reasoning with actions. In U. H. Engberg, K. G. Larsen, and P. D. Mosses, editors, *Proc. 6th Nordic Workshop on Programming Theory (Aarhus, 17–19 October, 1994)*, number NS-94-6 in BRICS Notes Series, pages 251–265, Dept. of Computer Science, Univ. of Aarhus, 1994.
- [18] I. A. Mason, S. Smith, and C. L. Talcott. From Operational Semantics to Domain Theory, 1994. Submitted to *Information and Computation*.
- [19] I. A. Mason and C. L. Talcott. Equivalence in functional languages with effects. *Journal of Functional Programming*, 1(3):297–327, 1991.
- [20] R. Milner. Operational and algebraic semantics of concurrent processes. In *Handbook of Theoretical Computer Science*. Elsevier, Amsterdam, 1990.
- [21] P. D. Mosses. Abstract semantic algebras! In D. Bjørner, editor, *Formal Description of Programming Concepts II*. IFIP, 1983.
- [22] P. D. Mosses. *Action Semantics*. Number 26 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1992.
- [23] P. D. Mosses, editor. *Proceedings of the First International Workshop on Action Semantics (Edinburgh, Scotland, April 1994)*, number NS-94-1 in BRICS Notes Series. Dept. of Computer Science, Univ. of Aarhus, 1994.
- [24] P. D. Mosses. Unified algebras and abstract syntax. In *Recent Trends in Data Type Specification*, volume 785 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.
- [25] P. D. Mosses and D. A. Watt. Pascal action semantics, Version 0.6, Mar. 1993.
- [26] H. Moura. *Action Notation Transformations*. Ph.D. thesis, University of Glasgow, 1993.
- [27] H. Moura and D. Watt. Action transformations in the ACTRESS compiler generator. In *CC'94*, volume 786 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.
- [28] C.-H. L. Ong. Non-determinism in a functional setting. In *8th Annual Symposium on logic in computer science*. IEEE, 1993.
- [29] P. Ørbæk. OASIS: An optimizing action-based compiler generator. In P. Fritzon, editor, *Proceedings of the 1994 Conference on Compiler Construction, Edinburgh*, volume 786 of LNCS, pages 1–15. Springer-Verlag, April 1994.

- [30] J. Palsberg. *Provably Correct Compiler Generation*. PhD thesis, Dept. of Computer Science, Univ. of Aarhus, 1992.
- [31] D. Park. Concurrency and automata on infinite sequences. In *Theoretical Computer Science*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, 1981.
- [32] L. C. Paulson. Co-induction and co-recursion in higher-order logic. Tech. report 304, University of Cambridge Computer Laboratory, 1994.
- [33] G. D. Plotkin. LCF considered as a programming language. *Theoretical Comput. Sci.*, 5:223–255, 1977.
- [34] I. Ulidowski. Equivalences on observable processes. In *Seventh Annual Symposium on Logic in Computer Science*. IEEE, 1992.
- [35] D. J. Walker. Bisimulation and divergence. *Information and Computation*, 85(2):202–241, 1990.

Index

- [], 7
- Id , 28
- \approx , 20
- \approx_{impl} , 20
- \approx'_{must} , 40
- \approx_m , 20
- $\hat{\approx}_m$, 17
- \approx'_{must} , 40
- \approx_m , 14
- \approx , 12
- \sqcap_{impl} , 13
- \sqcap'_{must} , 40
- \sqcap_m , 12
- \perp_m , 13
- \lesssim_{impl} , 29
- \lesssim'_{must} , 40
- \lesssim_m , 28
- \Downarrow , 9
- \Downarrow' , 40
- \ll_m , 11
- \downarrow_m , 11
- $\downarrow'_{\text{must}}$, 40
- \mapsto , 8
- \oplus , 14
- $\langle - \rangle_{\text{impl}}^{\text{hybr}}$, 31
- $\langle - \rangle_{\text{impl}}$, 29
- $\langle - \rangle_m$, 28
- \sqcap_m , 14
- \sqcup_{may} , 14
- \top_m , 13
- \circ , 16
- @, 5
- AN, 4
- and, 5
- and then, 5
- AS, 1
- ASD, 1
- C , 9
- C_{stuck} , 40
- chaos, 14
- check, 4
- complete, 4
- diverge, 13
- E , 8
- enact, 4
- escape, 4
- F , 10
- fail, 4
- m simulation, 28
- may, 11
- must, 11
- or, 5
- stuck, 39
- succeed, 13
- terminate, 13
- trap, 5
- unfold, 5
- unfolding, 5

Recent Publications in the BRICS Report Series

- RS-95-25 Søren B. Lassen. *Basic Action Theory*. May 1995. 47 pp.
- RS-95-24 Peter Ørbæk. *Can you Trust your Data?* April 1995. 15 pp. Appears in Mosses, Nielsen, and Schwartzbach, editors, *Theory and Practice of Software Development. 6th International Joint Conference CAAP/FASE, TAPSOFT '95 Proceedings*, LNCS 915, 1995, pages 575–590.
- RS-95-23 Allan Cheng and Mogens Nielsen. *Open Maps (at) Work*. April 1995. 33 pp.
- RS-95-22 Anna Ingólfssdóttir. *A Semantic Theory for Value-Passing Processes, Late Approach, Part II: A Behavioural Semantics and Full Abstractness*. April 1995. 33 pp.
- RS-95-21 Jesper G. Henriksen, Ole J. L. Jensen, Michael E. Jørgensen, Nils Klarlund, Robert Paige, Theis Rauhe, and Anders B. Sandholm. *MONA: Monadic Second-Order Logic in Practice*. May 1995. 17 pp.
- RS-95-20 Anders Kock. *The Constructive Lift Monad*. March 1995. 18 pp.
- RS-95-19 François Laroussinie and Kim G. Larsen. *Compositional Model Checking of Real Time Systems*. March 1995. 20 pp.
- RS-95-18 Allan Cheng. *Complexity Results for Model Checking*. February 1995. 18pp.
- RS-95-17 Jari Koistinen, Nils Klarlund, and Michael I. Schwartzbach. *Design Architectures through Category Constraints*. February 1995. 19 pp.
- RS-95-16 Dany Breslauer and Ramesh Hariharan. *Optimal Parallel Construction of Minimal Suffix and Factor Automata*. February 1995. 9 pp.
- RS-95-15 Devdatt P. Dubhashi, Grammati E. Pantziou, Paul G. Spirakis, and Christos D. Zaroliagis. *The Fourth Moment in Luby's Distribution*. February 1995. 10 pp. To appear in *Theoretical Computer Science*.
- RS-95-14 Devdatt P. Dubhashi. *Inclusion-Exclusion⁽³⁾ Implies Inclusion-Exclusion⁽ⁿ⁾*. February 1995. 6 pp.