LOGIC JOURNAL of the IGPL

Volume 9 Number 3 May 2001

Interest Group in Pure and Applied Logics

Editor-in-Chief:
DOV M. GABBAY

Executive Editors:
RUY de QUEIROZ
and
HANS JÜRGEN OHLBACH

Editorial Board:
Jon Barwise (deceased)
Wilfrid Hodges
Hans Kamp
Robert Kowalski
Grigori Mints
Ewa Orlowska
Amir Pnueli
Vaughan Pratt
Saharon Shelah
Johan van Benthem

OXFORD UNIVERSITY PRESS

ISSN 1367-0751

Subscription Information

Volume 9, 2001 (bimonthly) Full: Europe pounds sterling 275; Rest of World US$ 450. Personal: pounds sterling 138 (US$ 225). Please note that personal rates apply only when copies are sent to a private address and payment is made by a personal cheque or credit card.

Order information

Subscriptions can be accepted for complete volumes only. Prices include air-speeded delivery to Australia, Canada, India, Japan, New Zealand, and the USA. Delivery elsewhere is by surface post. Payment is required with all orders and may be made in the following ways:

Cheque (made payable to Oxford University Press)
National Girobank (account 500 1056)
Credit card (Access, Visa, American Express, Diners Club)
UNESCO Coupons
Bankers: Barclays Bank plc, PO Box 333, Oxford, UK. Code 20-65-18, Account 00715654.

Requests for sample copies, subscription enquiries, orders and changes of address should be sent to the Journals Subscriptions Department, Oxford University Press, Great Clarendon Street, Oxford OX2 6DP, UK. Tel: 01865 267907. Fax: 01865 267485.

Advertisements

Advertising enquiries should be addressed to Peter Carpenter, PRC Associates, The Annexe, Fitznells Manor, Chessington Road, Ewell Village, Surrey KT17 1TF, UK. Tel: 0181 786 7376. Fax: 0181 786 7262.

Copyright

©Oxford University Press 2001. All rights reserved: no part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without either the prior written permission of the Publishers, or a licence permitting restricted copying issued in the UK by the Copyright Licensing Agency Ltd, 90 Tottenham Court Road, London W1P 9HE, or in the USA by the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923.

Logic Journal of the IGPL (ISSN 1367-0751) is published bimonthly in January, March, May, July, September and November by Oxford University Press, Oxford, UK. Annual subscription price is US$ 450.00. *Logic Journal of the IGPL* is distributed by M.A.I.L. America, 2323 Randolph Avenue, Avenel, NJ 07001. Periodical postage paid at Rahway, New Jersey, USA and at additional entry points.
US Postmasters: Send address changes to *Logic Journal of the IGPL*, c/o Mercury International, 365 Blair Road, Avenel, NJ 07001, USA.

Back Issues

The current plus two back volumes are available from Oxford University Press. Previous volumes can be obtained from Dawsons Back Issues, Cannon House, Park Farm Road, Folkestone, Kent CT19 5EE, UK. Tel: +44 (0) 1303 203612. Fax: +44 (0) 1303 203617.

Logic Journal of the IGPL

Volume 9, Number 3, May 2001

Contents

Editorial 359
F. Kamareddine

Original Articles

The rewriting calculus — Part I 363
H. Cirstea and C. Kirchner

The rewriting calculus — Part II 401
H. Cirstea and C. Kirchner

Tableau Reasoning and Programming with Dynamic First Order Logic 435
J. van Eijck, J. Heguiabehere and B. Ó Nualláin

Theorem Proving in Infinitesimal Geometry 471
J. D. Fleuriot

A Simple Formalization and Proof for the Mutilated Chess Board 499
L. C. Paulson

Please visit the journal's World Wide Web site at
http://www.jigpal.oupjournals.org

Logic Journal of the Interest Group in Pure and Applied Logics

Editor-in-Chief:

Dov Gabbay
Department of Computer Science
King's College
Strand
London WC2R 2LS, UK
dg@dcs.kcl.ac.uk
Tel +44 20 7848 2930
Fax +44 20 7240 1071

Executive Editors:

Ruy de Queiroz
Departamento de Informática
UFPE em Recife
Caixa Postal 7851
Recife, PE 50732-970, Brazil
ruy@di.ufpe.br

Hans Jürgen Ohlbach
Inst. für Informatik
Ludwig-Maximilians-Universität
Öttingenstr. 67
D-80538 München
ohlbach@informatik.uni-muenchen.de
Tel +49 2178 2200
Fax +49 2178 2211

Editorial Board:

Jon Barwise (deceased)
Wilfrid Hodges, QMW, UK
Hans Kamp, Stuttgart, Germany
Robert Kowalski, ICSTM, UK
Grigori Mints, Stanford, USA
Ewa Orlowska, Warsaw, Poland
Amir Pnueli, Weizmann, Israel
Vaughan Pratt, Stanford, USA
Saharon Shelah, Jerusalem
Johan van Benthem,
ILLC, Amsterdam

Scope of the Journal

The *Journal* is the official publication of the International Interest Group in Pure and Applied Logics (IGPL), which is sponsored by The European Foundation for Logic, Language and Information (FoLLI), and currently has a membership of over a thousand researchers in various aspects of logic (symbolic, computational, mathematical, philosophical, etc.) from all over the world.

The *Journal* is published in hardcopy and in electronic form six times per year. Publication is fully electronic: submission, refereeing, revising, typesetting, checking proofs, and publishing, all is done via electronic mailing and electronic publishing.

Papers are invited in all areas of pure and applied logic, including: pure logical systems, proof theory, model theory, recursion theory, type theory, nonclassical logics, nonmonotonic logic, numerical and uncertainty reasoning, logic and AI, foundations of logic programming, logic and computation, logic and language, and logic engineering.

The *Journal* is an attempt to solve a problem in the logic (in particular, IGPL) community:

- Long delays and large backlogs in publication of papers in current journals.
- Very tight time and page number limits on submission.

Papers in the final form should be in LaTeX. The review process is quick, and is made mainly by other IGPL members.

Submissions

Submissions are made by sending a submission letter to the e-mail address: jigpl@dcs.kcl.ac.uk, giving the title and the abstract of the paper, and informing:

(i) of how to obtain the file electronically, if you have the .dvi or .ps file; or

(ii) that you have put the file (.dvi, .ps or .tex) in the public area ftp.dcs.kcl.ac.uk (137.73.8.10), directory, pub/jigpl/submissions

or, by sending 5 (five) hardcopies of the paper to the Editor-in-Chief.

URL: http://www.jigpal.oupjournals.org

Editorial

The 20th century gave birth to a computer technology that has dominated our lives. Such technology may be expensive to build and/or human lives may depend on it. We have overwhelming evidence from just under a century's work that the right logic and the right notion of symbolic manipulation (rewriting) can guarantee the safety and correctness of this technology saving money, and human lives and efforts. For this reason, we have seen and will continue to see new different logics and rewriting systems, extensions of old systems and the study of their theory and applications will be as thrive as it was in the last century. This is not surprising because the twentieth century was indeed a *century of complexity* and this complexity will be carried to this century. This complexity of information, the increasing interdependency of systems, the faster and more automatic travel of information, and the disastrous consequences of failure, lead to the need for establishing **Correctness**. Moreover, modern technological systems are just too complicated for humans to reason about unaided, so **automation** is needed. Furthermore, because modern systems have so many possible states, testing is often impractical. It seems that **proofs** are needed to cover infinitely many situations. The last century is evidence that formalisms needed to *aid in design* and to *ensure safety* must accommodate some *rewriting* and *automatic* search for and checking of proofs. These ideas were present long before the 20th century. In fact, Leibniz (1646–1717) conceived of *automated deduction*, when he wanted to find:

- a language L in which arbitrary concepts could be formulated, and
- a machine to determine the correctness of statements in L.

Such a machine can not work for every statement according to Gödel and Turing. Nevertheless, the need for automation has been overwhelming and its exploration in both the safe grounds and the dangerous borderlines continues to be challenging. The relevance of rewriting and automation is witnessed by the number of international conferences and events devoted to the subject. We cannot mention all these events and refer to the usual references. This volume however, is a selection of various papers that were presented at a collection of events on rewriting, automation and theorem proving that took place in year 2000 and were funded by different sources including: the European Union's IHP High Level Scientific Conferences support, the European Educational Forum, the UK Engineering and Physical Research Council EPSRC, the Royal Society and the Dutch research council NWO. The support of all these sources is greatly appreciated. These events are as follows:

- Winter Workshop in Logics, Types and Rewriting '00 on 2 February 2000.
  See `http://www.cee.hw.ac.uk/~fairouz/inaugural-workshop2000/`
- The EEF Foundations School in Deduction and Theorem Proving'00 on 6-16 April 2000.
  See `http://www.cee.hw.ac.uk/~fairouz/ukiischool2000/ukiischool.html`
- Festival Workshop in Foundations and Computing, FC'00 on 17-18 July 2000.
  See `http://www.cee.hw.ac.uk/~fairouz/festival/workshop1/`

Due to the succes of the above events, it was decided that a special issue should be published on the above themes. Some of the lecturers and speakers agreed to write

their material as articles for this volume. Of the submitted articles, five were selected for this volume.

The article of Cirstea and Kirchner is in two parts and is concerned with a new calculus called the ρ-calculus. The characteristic feature of the ρ-calculus is that it has an operator $\to$ used to build abstractions as in the λ-calculus as for instance $x \to x$ for the identity. Abstractions can also contain patterns as in first-order rewriting. For instance the rewrite rule $a \to b$ is in the ρ-calculus represented by the abstraction $a \to b$. The application of an abstraction to an argument is as in the λ-calculus, but now denoted by for instance $[x \to x](y)$ for the identity applied to a variable y. If the pattern of the left-hand side of the abstraction is not present in its argument, the application is rewritten to $\emptyset$, representing failure. For instance $[a \to b](b) \longrightarrow \emptyset$. If the pattern of the left-hand side is present in the argument, then the application is rewritten to the set consisting of the corresponding right-hand side. For instance, we have $[x \to x](y) \longrightarrow \{y\}$ and $[a \to b](a) \longrightarrow \{b\}$. Also sets consisting of more elements are used to represent non-determinism.

In the first part, the calculus is introduced and motivated and its syntax and evaluation rules for any theory are presented. Then, the encoding of the λ-calculus is presented and a discussion of confluence is given. In the second part, conditional rewriting is encoded and the calculus is extended with a *first* operator whose purpose is to detect rule application failure. This extension enables the encoding of strategy based rewriting processes and is used to give an operational semantics to ELAN which is an environment for specifying and prototyping deduction systems in a language based on labelled conditional rules and strategies to control rule application.

The article of Jan van Eijck and Juan Heguiabehere and Breanndán Ó Nualláin presents a tableau system for dynamic first-order logic (DFOL for short), a formalism originally introduced by Groenendijk and Stokhof to account for certain aspects of natural language semantics and anaphora. The language presented in this paper contains explicit substitutions and the choice operator $\cup$. The language is further extended with the finite iteration *-operator (DFOL*). Soundness and completeness of the tableau method for DFOL and then DFOL* is proved. The authors illustrate through significant examples the usefulness of DFOL and DFOL* and of the related tableau method to represent program execution and to derive pre/post conditions in the style of Hoare logic. They also show the potential benefit of their tableau method as a tool in computational semantics of natural language.

The article of Jacques Fleuriot, reports the formalisation in the theorem prover Isabelle of a theory of non-standard geometry based on infinitely small and large reals. The theory is based on so-called hyperreal vectors which are sequences of real vectors with two such sequences being equal if they coincide on an element in an ultrafilter (an abstract way to express that they are equal almost everywhere). The paper uses the full power of the Isabelle-HOL formalism in order to get a smooth development. It can be seen as a reference paper on the basis of infinitesimal geometry. As mentioned by the author, extending usual operations to infinitely small or large objects is very subtle and can easily be done the wrong way. The fact that the theory is completely developed in Isabelle-HOL is consequently really useful.

The article of Paulson presents a short and natural mechanisation of the proof of the mutilated chessboard problem in Isabelle. This exercise is used to demonstrate some important principles in the manipulation of systems of this kind. Particular emphasis

is put on the use of inductive definitions. These are of interest both because they allow the user to give intuitive definitions of e.g. what is a domino and what is a tiling, and because they inherently capture the essence of the concepts being formalised (therefore, for example, no erroneous tiling can be generated in the development of a proof). Moreover, Isabelle's tactics technology, together with the conciseness offered by inductive definitions, makes it possible to derive a formalisation that is much shorter than in similar works based on other provers.

FAIROUZ KAMAREDDINE

The rewriting calculus — Part I

HORATIU CIRSTEA, *LORIA and INRIA, Campus Scientifique, BP 239, 54506 Vandoeuvre-lès-Nancy, France. E-mail: Horatiu.Cirstea@loria.fr.*

CLAUDE KIRCHNER, *LORIA and INRIA, Campus Scientifique, BP 239, 54506 Vandoeuvre-lès-Nancy, France. E-mail: Claude.Kirchner@loria.fr.*

Abstract

The ρ-calculus integrates in a uniform and simple setting first-order rewriting, λ-calculus and non-deterministic computations. Its abstraction mechanism is based on the rewrite rule formation and its main evaluation rule is based on matching modulo a theory T.

In this first part, the calculus is motivated and its syntax and evaluation rules for any theory T are presented. In the syntactic case, *i.e.* when T is the empty theory, we study its basic properties for the untyped case. We first show how it uniformly encodes λ-calculus as well as first-order rewriting derivations. Then we provide sufficient conditions for ensuring confluence of the calculus.

Keywords: rewriting, strategy, non-determinism, matching, rewriting-calculus, lambda-calculus, rule based language.

1 Introduction

1.1 Rewriting, computer science and logic

It is a common claim that rewriting is ubiquitous in computer science and mathematical logic. And indeed the rewriting concept appears from the very theoretical settings to the very practical implementations. Some extreme examples are the mail system under Unix that uses rules in order to rewrite mail addresses in canonical forms (see the `/etc/sendmail.cf` file in the configuration of the mail system) and the transition rules describing the behaviors of tree automata. Rewriting is used in semantics in order to describe the meaning of programming languages [31] as well as in program transformations like, for example, re-engineering of Cobol programs [54]. It is used in order to compute [12], implicitly or explicitly as in Mathematica [59], MuPAD [42] or OBJ [22], but also to perform deduction when describing by inference rules a logic [23], a theorem prover [28] or a constraint solver [29]. It is of course central in systems making the notion of rule an explicit and first class object, like expert systems, programming languages based on equational logic [45], algebraic specifications (*e.g.* OBJ [22]), functional programming (*e.g.* ML [40]) and transition systems (*e.g.* Murphi [11]).

It is hopeless to try to be exhaustive and the cases we have just mentioned show part of the huge diversity of the rewriting concept. When one wants to focus on the underlying notions, it becomes quickly clear that several technical points should be settled. For example, what kind of objects are rewritten? Terms, graphs, strings, sets,

 ©Oxford University Press

multisets, others? Once we have established this, what is a rewrite rule? What is a left-hand side, a right-hand side, a condition, a context? And then, what is the effect of a rule application? This leads immediately to defining more technical concepts like variables in bound or free situations, substitutions and substitution application, matching, replacement; all notions being specific to the kind of objects that have to be rewritten. Once this is solved one has to understand the meaning of the application of a set of rules on (classes of) objects. And last but not least, depending on the intended use of rewriting, one would like to define an induced relation, or a logic, or a calculus.

In this very general picture, we introduce a calculus whose main design concept is to make all the basic ingredients of rewriting explicit objects, in particular the notions of rule *application* and *result*. We concentrate on *term* rewriting, we introduce a very general notion of rewrite rule and we make the rule application and result explicit concepts. These are the basic ingredients of the *rewriting-* or *ρ-calculus* whose originality comes from the fact that terms, rules, rule application and therefore rule application strategies are all treated at the object level.

1.2 How does the rewriting calculus work?

In ρ-calculus we can explicitly represent the application of a rewrite rule, as for example $2 \to s(s(0))$, to a term, *e.g.* the constant 2, as the object $[2 \to s(s(0))](2)$ which evaluates to the singleton $\{s(s(0))\}$. This means that the rule application binary symbol "$[_](_)$" is part of the calculus syntax.

As we have seen a rule application can be reduced to a singleton, but it may also fail as in $[2 \to s(s(0))](3)$ that evaluates to the empty set $\emptyset$, or it can be reduced to a set with more than one element as exemplified later in this section and explained in Section 2.4. Of course, variables may be used in rewrite rules as in $[x+0 \to x](4+0)$. In this last case the evaluation mechanism of the calculus will reduce the application to $\{4\}$. In fact, when evaluating this expression, the variable x is bound to 4 via a mechanism classically called matching, and the result of the evaluation is obtained by instantiating accordingly the variable x from the right hand side of the rewrite rule. We recover, thus, the classical way term rewriting is acting.

Where this game becomes even more interesting is that "$_ \to _$", the rewrite binary operator, is integrally part of the calculus syntax. This is a powerful abstraction operator whose relationship with λ-abstraction [7] could provide a useful intuition: A λ-expression $\lambda x.t$ can be represented in the ρ-calculus as the rewrite rule $x \to t$. Indeed, the β-redex $(\lambda x.t\ u)$ is nothing else than $[x \to t](u)$ (*i.e.* the application of the rewrite rule $x \to t$ to the term u) which reduces to $\{\{x/u\}t\}$ (*i.e.* the application of the substitution $\{x/u\}$ to the term t).

We are aware of other ways to abstract on terms or patterns in lambda-calculus e.g. the works of Colson, Kesner, van Oostrom [10, 57, 32] or Peyton-Jones [50]. For example, the λ-calculus with patterns presented in [50] can be given a direct representation in the ρ-calculus. Let us consider, for example, the λ-term $\lambda(PAIR\ x\ y).x$ that selects the first element of a pair and the application $\lambda(PAIR\ x\ y).x\ (PAIR\ a\ b)$ that evaluates to a. The representation in the ρ-calculus of the first λ-term is $PAIR(x,y) \to x$ and the corresponding application $[PAIR(x,y) \to x](PAIR(a,b))$ ρ-evaluates to $\{\{x/a, y/b\}x\}$, that is to $\{a\}$.

Of course we have to make clear what a substitution $\{x/u\}$ is and how it applies to a term. But there is no surprise here and we consider a substitution mechanism that preserves the correct variable bindings via the appropriate α-conversion. In order to make this point clear in the paper, as in [13], we will make a strong distinction between *substitution* (which takes care of variable binding) and *grafting* (that performs replacement directly).

When building abstractions, *i.e.* rewrite rules, there is a priori no restriction. A rewrite rule may introduce new variables as in the rule $f(x) \to g(x,y)$ that when applied to the term $f(a)$ evaluates to $\{g(a,y)\}$, leaving the variable y free. It may also rewrite an object into a rewrite rule as in the application $[x \to (f(y) \to g(x,y))](a)$ that evaluates to the singleton $\{f(y) \to g(a,y)\}$. In this case the variable x is free in the rewrite rule $f(y) \to g(x,y)$ but is bound in the rule $x \to (f(y) \to g(x,y))$. More generally, the object formation in ρ-calculus is unconstrained. Thus, the application of the rule $b \to c$ after the rule $a \to b$ to the term a is written $[b \to c]([a \to b](a))$ and as expected the evaluation mechanism will produce first $[b \to c](\{b\})$ and then $\{c\}$. It also allows us to make use in an explicit and direct way of non-terminating or non-confluent (equational) rewrite systems. For example the application of the rule $a \to a$ to the term a $([a \to a](a))$ terminates, since it is applied only once and does not represent the *repeated* application of the rewrite rule $a \to a$.

So, basic ρ-calculus objects are built from a signature, a set of variables, the abstraction operator "$\to$", the application operator "[ ]( )", and we consider sets of such objects. This gives to the ρ-calculus the ability to handle non-determinism in the sense of sets of results. This is achieved via the explicit handling of reduction result sets, including the empty set that records the fundamental information of rule application failure. For example, if the symbol $+$ is assumed to be commutative then $x + y$ is equivalent modulo commutativity to $y + x$ and thus applying the rule $x+y \to x$ to the term $a+b$ results in $\{a, b\}$. Since there are two different ways to apply (match) this rewrite rule modulo commutativity the result is a set that contains two different elements corresponding to two possibilities. This ability to integrate specific computations in the matching process allows us for example to use the ρ-calculus for deduction modulo purposes as proposed in [14].

To summarize, in ρ-calculus abstraction is handled via the arrow binary operator, matching is used as the parameter passing mechanism, substitution takes care of variable bindings and results sets are handled explicitly.

1.3 Rewriting relation versus rewriting calculus

A ρ-calculus term contains all the (rewrite rule) information needed for its evaluation. This is also the case for λ-calculus but it is quite different from the usual way term rewrite *relations* are defined.

The rewrite relation generated by a rewrite system $\mathcal{R} = \{l_1 \to r_1, \ldots, l_n \to r_n\}$ is defined as the smallest transitive relation stable by context and substitution and containing $(l_1, r_1), \ldots, (l_n, r_n)$. For example if $\mathcal{R} = \{a \to f(a)\}$, then the rewrite relation contains $(a, f(a))$, $(a, f(f(a)))$, $(f(a), f(f(a))), \ldots$ and one says that the derivation $a \to f(a) \to f(f(a)) \to \ldots$ is generated by $\mathcal{R}$.

In ρ-calculus the situation is different since ρ-evaluation will reduce a given ρ-term in which all the rewriting information is explicit. It is customary to say that the rewrite

system $a \to a$ is not terminating because it generates the derivation $a \to a \to a \to \dots$. In ρ-calculus the same infinite derivation should be explicitly built (for example using an iterator) and all the evaluation information should be present in the starting term as in $[a \to a]([a \to a]([a \to a](a)))$ whose evaluation corresponds to the three steps derivation $a \to a \to a \to a$.

There is thus a big difference between the way one can define rewrite derivations generated by a rewrite system and their representation in ρ-calculus: in the first case the derivation construction is implicit and left at the meta-level, in the later case, all rewrite steps should be explicitly built.

1.4 Integration of first-order rewriting and higher-order logic

We are introducing a new calculus in a heavily-charged landscape. Why one more? There are several complementary answers that we will make explicit in this work. One of them is the unifying principle of the calculus with respect to algebraic and higher-order theories.

The integration of first-order and higher-order paradigms has been one of the main problems raised since the beginning of the study of programming language semantics and of proof environments. The λ-calculus emerged in the thirties and had a deep influence on the development of theoretical computer-science as a simple but powerful tool for describing programming language semantics as well as proof development systems. Term rewriting for its part emerged as an identified concept in the late sixties and it had a deep influence in the development of algebraic specifications as well as in theorem proving.

Because the two paradigms have a lot in common but have extremely useful complementary properties, many works address the integration of term rewriting with λ-calculus. This has been handled either by enriching first-order rewriting with higher-order capabilities or by adding to λ-calculus algebraic features allowing one, in particular, to deal with equality in an efficient way. In the first case, we find the works on CRS [38], XRS [49] and other higher-order rewriting systems [58, 44], in the second case the works on combination of λ-calculus with term rewriting [46, 5, 21, 30] to mention only a few.

Our previous works on the control of term rewriting [35, 56, 3] led us to introduce the ρ-calculus. Indeed we realized that the tool that is needed in order to control rewriting should be made explicit and could be itself naturally described using rewriting. By viewing the arrow rewrite symbol as an abstraction operator, we strictly generalize the abstraction mechanism of λ-calculus, by making the rule application explicit, we get full control of the rewrite mechanism and as a consequence we obtain with the ρ-calculus a uniform integration of algebraic computation and λ-calculus.

1.5 Basic properties and uses of the ρ-calculus

One of the main properties of the calculus we are concentrating on is confluence. We will see that the ρ-calculus is not confluent in the general case. The use of sets for representing the reduction results is the main cause of non-confluence. This comes from the fact that in the definition of a standard rewrite step, a rule is applied only when a successful match is found and in this case the reduced term exists and is unique

(even if several matches exist). In ρ-calculus we are in a very different situation since a rule application always yields a unique result consisting either of a non-empty set representing all the possible reduced terms (one per different match) or of an empty set representing the impossibility to apply a standard rewrite step.

The confluence can be recovered if the evaluation rules of ρ-calculus are guided by an appropriate strategy. This strategy should first handle properly the problems related to the propagation of failure over the operators of the calculus. It should also take care of the correct handling of sets with more than one element in non-linear contexts. We are presenting this strategy whose full details are given in [8].

We will see that the ρ-calculus can be used for representing some simpler calculi as λ-calculus and rewriting even in the conditional case. This is achieved by restricting the syntax and the evaluation rules of the ρ-calculus in order to represent the terms of the two calculi. We then show that for any reduction in the λ-calculus or term rewriting, a corresponding natural reduction in the ρ-calculus can be found.

1.6 Structure of this work and paper

The presentation of this work is divided in two parts, the second one being called hereafter *Part II*.

The purpose of this first part is to introduce the ρ-calculus, its syntax and evaluation rules and to show how it can be used in order to naturally encode λ-calculus and standard term rewriting. We also show in *Part II*, and indeed this was our first motivation, that it can be used to encode conditional rewriting and that it provides a semantics for the rewrite based language ELAN.

In the next section, we introduce the general ρ_T-calculus, where T is a theory used to internalize specific knowledge like associativity and commutativity of certain operators. We present the syntax of the calculus, its evaluation rules together with examples. We emphasize in particular the important role of the matching theory T. We show in Section 3 how ρ-calculus can be used to encode in a uniform way term rewriting and λ-calculus. Then, in Section 4, we restrict to the $\rho_\emptyset$-calculus (also shortly denoted ρ-calculus), the calculus where only syntactic matching is allowed (*i.e.* the theory T is assumed to be the trivial one), and we present the confluence properties of this calculus. We assume the reader familiar with the standard notions of term rewriting [16, 36, 4, 33] and with the basic notions of λ-calculus [2]. For the basic concepts about rule based constraint solving and *deduction modulo*, we refer respectively to [29, 37] and [14].

2 Definition of the ρ_T-calculus

We assume given in this section a theory T defined equationally or by any other means.

A calculus is defined by the following five components:

1. First its *syntax* that makes precise the formation of the objects manipulated by the calculus as well as the formation of substitutions that are used by the evaluation mechanism. In the case of ρ_T-calculus, the core of the object formation relies on a first-order signature together with rewrite rules formation, rule application and

sets of results.

2. The description of the *substitution application* to terms. This description is often given at the meta-level, except for explicit substitution frameworks. For the description of the ρ_T-calculus that we give here, we use (higher-order) substitutions and not grafting, *i.e.* the application takes care of variable bindings and therefore uses α-conversion.
3. The *matching algorithm* used to bind variables to their actual values. In the case of ρ_T-calculus, this is matching modulo the theory T. In practical cases it will be higher-order-pattern matching, or equational matching, or simply syntactic matching or combination of any of these. The matching theory is specified as a parameter (the theory T) of the calculus and when it is clear from the context this parameter is omitted.
4. The *evaluation rules* describing the way the calculus operates. It is the glue between the previous components. The simplicity and clarity of these rules are fundamental for its usability.
5. The *strategy* guiding the application of the evaluation rules. Depending on the strategy employed we obtain different versions and therefore different properties for the calculus.

This section makes explicit all these components for the ρ_T-calculus and comments our main choices.

2.1 Syntax of the ρ_T-calculus

Definition 2.1 We consider $\mathcal{X}$ a set of variables and $\mathcal{F} = \bigcup_m \mathcal{F}_m$ a set of ranked function symbols, where for all m, $\mathcal{F}_m$ is the subset of function symbols of arity m. We assume that each symbol has a unique arity *i.e.* that the $\mathcal{F}_m$ are disjoint. We denote by $\mathcal{T}(\mathcal{F},\mathcal{X})$ the set of first-order terms built on $\mathcal{F}$ using the variables in $\mathcal{X}$. The set of basic ρ-terms, denoted $\varrho(\mathcal{F},\mathcal{X})$, is the smallest set of objects formed according to the following rules:

- the variables in $\mathcal{X}$ are ρ-terms,
- if $t_1,\dots,t_n$ are ρ-terms and $f \in \mathcal{F}_n$ then $f(t_1,\dots,t_n)$ is a ρ-term,
- if $t_1,\dots,t_n$ are ρ-terms then $\{t_1,\dots,t_n\}$ is a ρ-term (the empty set is denoted $\emptyset$),
- if t and u are ρ-terms then $[t](u)$ is a ρ-term (application),
- if t and u are ρ-terms then $t \to u$ is a ρ-term (abstraction or rewrite rule).

The set of basic ρ-terms can thus be inductively defined by the following grammar:

$$\rho\text{-terms} \quad t \quad ::= \quad x \mid f(t,\dots,t) \mid \{t,\dots,t\} \mid [t](t) \mid t \to t$$

where $x \in \mathcal{X}$ and $f \in \mathcal{F}$. Notice that this syntax does not make use of the theory T.

A term may be viewed as a *finite labeled ordered tree*, the leaves of which are labeled with variables or constants and the internal nodes of which are labeled with symbols of positive arity.

Definition 2.2 A *position* (also called *occurrence*) of a term (seen as a tree) is represented as a sequence ω of positive integers describing the path from the root of t to

the root of the sub-term at that position. We denote by $t_{\lceil s \rceil_p}$ the term t containing the sub-term s at the position p. The symbol at the position p of a term t is denoted by $t(p)$.

We call *functional position* of a ρ-term t, any occurrence p of the term whose symbol belongs to $\mathcal{F}$, *i.e.* $t(p) \in \mathcal{F}$. The set of all positions of a term t is denoted by $\mathcal{P}os(t)$. The set of all functional positions of a term t is denoted by $\mathcal{FP}os(t)$.

The position of a sub-term in a set ρ-term is obtained by considering one of the possible tree representations of the respective ρ-term.

We adopt a very general discipline for the rewrite rule formation, and we do not enforce any of the standard restrictions often used in the term rewriting community like non-variable left-hand sides or occurrence of the right-hand side variables in the left-hand side. We also consider rewrite rules containing rewrite rules as well as rewrite rule application. For convenience, we consider that the symbols $\{\}$ and $\emptyset$ both represent the empty set. We usually use the notation f instead of $f()$ for a function symbol of arity 0 (*i.e.* a constant). For the terms of the form $\{t_1, \ldots, t_n\}$ we assume, as usually, that the comma is an associative, commutative and idempotent function symbol.

The main intuition behind this syntax is that a rewrite rule is an abstraction, the left-hand side of which determines the bound variables and some contextual information. Having new variables in the right-hand side is just the ability to have free variables in the calculus. We will come back to this later but to support the intuition let us mention that the λ-terms [2] and standard first-order rewrite rules [16, 4] are clearly objects of this calculus. For example, the λ-term $\lambda x.(y\ x)$ corresponds to the ρ-term $x \to [y](x)$ and a rewrite rule in first-order rewriting corresponds to the same rewrite rule in the rewriting-calculus.

We have chosen sets as the data structure for handling the potential non-determinism. A set of terms can be seen as the set of distinct results obtained by applying a rewrite rule to a term. Other choices could be made depending on the intended use of the calculus. For example, if we want to provide all the results of an application, including the identical ones, a multi-set could be used. When the order of the computation of the results is important, lists could be employed. Since in this presentation of the calculus we focus on the possible results of a computation and not on their number or order, sets are used. The confluence properties presented in Section 4 are preserved in a multi-set approach. It is clear that for the list approach only a confluence modulo permutation of lists can be obtained.

The following examples show the very expressive syntax that is allowed for ρ-terms.

Example 2.3 If we consider $\mathcal{F}_0 = \{a, b, c\}$, $\mathcal{F}_1 = \{f\}$, $\mathcal{F}_2 = \{g\}$, $\mathcal{F} = \mathcal{F}_0 \cup \mathcal{F}_1 \cup \mathcal{F}_2$ and x, y variables in $\mathcal{X}$, some ρ-terms from $\varrho(\mathcal{F}, \mathcal{X})$ are:

- $[a \to b](a)$; this denotes the application of the rewrite rule $a \to b$ to the term a. We will see that evaluating this application results in $\{b\}$.
- $[g(x, y) \to f(x)](g(a, b))$; a classical rewrite rule application.
- $[x \to x + y](a)$; a rewrite rule with a free variable y. We will see later why the result of this application is $\{a + y\}$ where the variable y remains free.
- $[y \to [x \to x + y](b)]([x \to x](a))$; a ρ-term that corresponds to the λ-term $(\lambda y.((\lambda x.x + y)\ b))\ ((\lambda x.x)\ a)$. In the rewrite rule $x \to x + y$ the variable y is free but in the rewrite rule $y \to [x \to x + y](b)$ this variable is bound.

- $[x \to [x](x)](x \to [x](x))$; the well-known $(\omega\omega)$ λ-term. We will see that the evaluation of this term is not terminating.
- $[[(x \to x+1) \to (1 \to x)](a \to a+1)](1)$; a more complicated ρ-term without corresponding standard rewrite rule or λ-term.

2.2 Grafting versus substitution

Since we are dealing with $\to$ as a binder, like for any calculus involving binders (as the λ-calculus), α-conversion should be used to obtain a correct substitution calculus and the first-order substitution (called here grafting) is not directly suitable for the ρ-calculus. We consider the usual notions of α-conversion and higher-order substitution as defined for example in [13].

This is the reason for introducing an appropriate notion of bound variables renaming in Definition 2.5. It computes a variant of a ρ-term which is equivalent modulo α-conversion to the initial term.

Definition 2.4 The set of free variables of a ρ-term t is denoted by $FV(t)$ and is defined by:

1. if $t = x$ then $FV(t) = \{x\}$,
2. if $t = f(u_1, \ldots, u_n)$ then $FV(t) = \bigcup_{i=1,\ldots,n} FV(u_i)$,
3. if $t = \{u_1, \ldots, u_n\}$ then $FV(t) = \bigcup_{i=1,\ldots,n} FV(u_i)$,
4. if $t = [u](v)$ then $FV(t) = FV(u) \cup FV(v)$,
5. if $t = u \to v$ then $FV(t) = FV(v) \setminus FV(u)$.

Definition 2.5 Given a set $\mathcal{Y}$ of variables, the application $\alpha_{\mathcal{Y}}$ (called α-conversion) is defined by:

- $\alpha_{\mathcal{Y}}(x) = x$,
- $\alpha_{\mathcal{Y}}(f(u_1, \ldots, u_n)) = f(\alpha_{\mathcal{Y}}(u_1), \ldots, \alpha_{\mathcal{Y}}(u_n))$,
- $\alpha_{\mathcal{Y}}(\{t\}) = \{\alpha_{\mathcal{Y}}(t)\}$,
- $\alpha_{\mathcal{Y}}([t](u)) = [\alpha_{\mathcal{Y}}(t)](\alpha_{\mathcal{Y}}(u))$,
- $\alpha_{\mathcal{Y}}(u \to v) = \alpha_{\mathcal{Y}}(u) \to \alpha_{\mathcal{Y}}(v)$, if $FV(u) \cap \mathcal{Y} = \emptyset$,
- $\alpha_{\mathcal{Y}}(u \to v) = (\{x_i \mapsto y_i\}_{x_i \in FV(u)}\ \alpha_{\mathcal{Y}}(u)) \to (\{x_i \mapsto y_i\}_{x_i \in FV(u)}\ \alpha_{\mathcal{Y}}(v))$,
  if $x_i \in FV(u) \cap \mathcal{Y}$ and y_i are "fresh" variables and where $\{x \mapsto y\}$ denotes the replacement of the variable x by the variable y in the term on which it is applied.

This allows us to define the usual substitution and grafting operations:

Definition 2.6 A *valuation* θ is a finite binding of the variables $x_1, \ldots, x_n$ to the terms $t_1, \ldots, t_n$, *i.e.* a finite set of couples $\{(x_1, t_1), \ldots, (x_n, t_n)\}$.

From a given valuation θ we can define the following two notions of substitution and grafting:

- the *substitution* extending θ is denoted $\Theta = \{x_1/t_1, \ldots, x_n/t_n\}$,
- the *grafting* extending θ is denoted $\bar{\Theta} = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$.

Θ and $\bar{\Theta}$ are structurally defined by:

- $\Theta(x) = u$, if $(x, u) \in \theta$
- $\Theta(f(t_1 \dots t_n)) = f(\Theta(t_1) \dots \Theta(t_n))$
- $\Theta(\{t_1, \dots, t_n\}) = \{\Theta(t_1), \dots, \Theta(t_n)\}$
- $\Theta([t](u)) = [\Theta(t)](\Theta(u))$
- $\Theta(u \to v) = \Theta(u') \to \Theta(v')$

- $\bar{\Theta}(x) = u$, if $(x, u) \in \theta$
- $\bar{\Theta}(f(t_1 \dots t_n)) = f(\bar{\Theta}(t_1) \dots \bar{\Theta}(t_n))$
- $\bar{\Theta}(\{t_1, \dots, t_n\}) = \{\bar{\Theta}(t_1), \dots, \bar{\Theta}(t_n)\}$
- $\bar{\Theta}([t](u)) = [\bar{\Theta}(t)](\bar{\Theta}(u))$
- $\bar{\Theta}(u \to v) = \bar{\Theta}(u) \to \bar{\Theta}(v)$

where we consider that z_i are fresh variables (*i.e.* $\theta z_i = z_i$), the z_i do not occur in u and v and for any $y \in FV(u)$, $z_i \notin FV(\theta y)$, and u', v' are defined by:
$u' = \{y_i \mapsto z_i\}_{y_i \in FV(u)}\ \alpha_{FV(u) \cup \mathcal{V}ar(\theta)}(u)$,
$v' = \{y_i \mapsto z_i\}_{y_i \in FV(u)}\ \alpha_{FV(u) \cup \mathcal{V}ar(\theta)}(v)$.
using the following notations: The set of variables $\{x_1, \dots, x_n\}$ is called the domain of the substitution Θ or of the grafting $\bar{\Theta}$ and is denoted by $\mathcal{D}om(\Theta)$ or $\mathcal{D}om(\bar{\Theta})$ respectively. The set of all the variables from Θ is $\mathcal{V}ar(\Theta) = \cup_{x \in \mathcal{D}om(\Theta)} \Theta(x) \cup \mathcal{D}om(\Theta)$.

Recall that $\{x_1/t_1, \dots, x_n/t_n\}$ is the simultaneous substitution of the variables $x_1, \dots, x_n$ by the terms $t_1, \dots, t_n$ and not the composition $\{x_1/t_1\} \dots \{x_n/t_n\}$.

There is nothing new in the definition of substitution and grafting except that the abstraction works here on terms and not only on variables. The burden of variable handling could be avoided by using an explicit substitution mechanism in the spirit of [6]. We sketched such an approach in [9] and this is detailed in [8].

2.3 Matching

Computing the matching substitutions from a ρ-term t to a ρ-term t' is an important parameter of the ρ_T-calculus. We first define matching problems in a general setting:

Definition 2.7 For a given theory T over ρ-terms, a *T-match-equation* is a formula of the form $t \ll^?_T t'$, where t and t' are ρ-terms. A substitution σ is a solution of the T-match-equation $t \ll^?_T t'$ if $T \models \sigma(t) = t'$. A *T-matching system* is a conjunction of T-match-equations. A substitution is a solution of a T-matching system P if it is a solution of all the T-match-equations in P. We denote by $\mathbf{F}$ a T-matching system without solution. A T-matching system is called *trivial* when all substitutions are solution of it.
We define the function *Solution* on a T-matching system $\mathcal{S}$ as returning the set of all T-matches of $\mathcal{S}$ when $\mathcal{S}$ is not trivial and $\{\mathbb{ID}\}$, where $\mathbb{ID}$ is the identity substitution, when $\mathcal{S}$ is trivial.

Notice that when the matching system has no solution the function *Solution* returns the empty set.

Since in general we could consider arbitrary theories over ρ-terms, T-matching is in general undecidable, even when restricted to first-order equational theories [29]. In order to overcome this undecidability problem, one can think of using constraints as in constrained higher-order resolution [26] or constrained deduction [34]. But we are interested here in the decidable cases. Among them we can mention higher-order-pattern matching that is decidable and unitary as a consequence of the decidability of pattern unification [41, 15], higher-order matching which is known to be decidable

up to the fourth order [47, 48, 17, 24] (the decidability of the general case being still open), many first-order equational theories including associativity, commutativity, distributivity and most of their combinations [43, 52].

For example when T is empty, the syntactic matching substitution from t to t', when it exists, is unique and can be computed by a simple recursive algorithm given for example by G. Huet [27]. It can also be computed by the following set of rules *SyntacticMatching* where $f, g \in \mathcal{F}$ and the symbol $\wedge$ is assumed to be associative and commutative.

Decomposition	$(f(t_1, \ldots, t_n) \ll^?_\emptyset f(t'_1, \ldots, t'_n)) \wedge P$	$\mapsto\!\!\!\!\!\rightarrow$	$\bigwedge_{i=1\ldots n} t_i \ll^?_\emptyset t'_i \wedge P$
SymbolClash	$(f(t_1, \ldots, t_n) \ll^?_\emptyset g(t'_1, \ldots, t'_m)) \wedge P$	$\mapsto\!\!\!\!\!\rightarrow$	$\mathbf{F}$ if $f \neq g$
MergingClash	$(x \ll^?_\emptyset t) \wedge (x \ll^?_\emptyset t') \wedge P$	$\mapsto\!\!\!\!\!\rightarrow$	$\mathbf{F}$ if $t \neq t'$
VariableClash	$(f(t_1, \ldots, t_n) \ll^?_\emptyset x) \wedge P$	$\mapsto\!\!\!\!\!\rightarrow$	$\mathbf{F}$ if $x \in \mathcal{X}$

FIG. 1. *SyntacticMatching* - Rules for syntactic matching

Proposition 2.8 The normal form by the rules in *SyntacticMatching* of any matching problem $t \ll^?_\emptyset t'$ exists and is unique. After removing from the normal form any duplicated match-equation and the trivial match-equations of the form $x \ll^?_\emptyset x$ for any variable x, if the resulting system is:

1. $\mathbf{F}$, then there is no match from t to t' and $\mathcal{S}olution(t \ll^?_\emptyset t') = \mathcal{S}olution(\mathbf{F}) = \emptyset$,
2. of the form $\bigwedge_{i \in I} x_i \ll^?_\emptyset t_i$ with $I \neq \emptyset$, then the substitution $\sigma = \{x_i/t_i\}_{i \in I}$ is the unique match from t to t' and $\mathcal{S}olution(t \ll^?_\emptyset t') = \mathcal{S}olution(\bigwedge_{i \in I} x_i \ll^?_\emptyset t_i) = \{\sigma\}$,
3. empty, then t and t' are identical and $\mathcal{S}olution(t \ll^?_\emptyset t) = \{\mathbb{ID}\}$.

PROOF. See [33].

□

Example 2.9 If we consider the matching problem $(h(x, g(x, y)) \ll^?_\emptyset h(a, g(a, b))$, first we apply the matching rule *Decomposition* and we obtain the system with the two match-equations $(x \ll^?_\emptyset a)$ and $(g(x, y) \ll^?_\emptyset g(a, b))$. When we apply the same rule once again for the second equation we obtain $(x \ll^?_\emptyset a)$ and $(y \ll^?_\emptyset b)$ and thus, the initial match-equation is reduced to the system $(x \ll^?_\emptyset a) \wedge (x \ll^?_\emptyset a) \wedge (y \ll^?_\emptyset b)$ and $\mathcal{S}olution(h(x, g(x, y)) \ll^?_\emptyset h(a, g(a, b)) = \{\{x/a, y/b\}\}$.

For the matching problem $(g(x, x) \ll^?_\emptyset g(a, b))$ we apply, as before, *Decomposition* and we obtain the system $(x \ll^?_\emptyset a) \wedge (x \ll^?_\emptyset b)$. This latter system is reduced by the matching rule *MergingClash* to $\mathbf{F}$ and thus, $\mathcal{S}olution(g(x, x) \ll^?_\emptyset g(a, b)) = \emptyset$.

This syntactic matching algorithm has an easy and natural extension when a symbol $+$ is assumed to be commutative. In this case, the previous set of rules should be

completed with

$$CommDec \quad (t_1 + t_2) \ll^?_{C(+)} (t'_1 + t'_2) \wedge P \quad \mapsto\!\!\!\!\rightarrow$$
$$((t_1 \ll^?_{C(+)} t'_1 \wedge t_2 \ll^?_{C(+)} t'_2) \vee (t_1 \ll^?_{C(+)} t'_2 \wedge t_2 \ll^?_{C(+)} t'_1)) \wedge P$$

where disjunction should be handled in the usual way. In this case of course the number of matches could be exponential in the size of the initial left-hand sides.

Example 2.10 When matching modulo commutativity the term $x+y$, with $+$ defined as commutative, against the term $a+b$, the rule $CommDec$ leads to

$$((x \ll^?_{C(+)} a \wedge y \ll^?_{C(+)} b) \vee (x \ll^?_{C(+)} b \wedge y \ll^?_{C(+)} a))$$

and thus, we obtain two substitutions as solution for the initial matching problem, *i.e.* $Solution(x + y \ll^?_{C(+)} a + b) = \{\{x/a, y/b\}, \{x/b, y/a\}\}$.

Matching modulo associativity-commutativity (AC) is often used. It could be defined either in a rule based way as in [1, 37] or in a semantic way as in [18]. A restricted form of associative matching called *list matching* is used in the ASF+SDF system [53]. In the Maude system any combination of the associative, commutative and idempotency properties is available [19].

2.4 Evaluation rules of the ρ_T-calculus

Assume we are given a theory T over ρ-terms having a decidable matching problem. The use of constraints would allow us to drop this last restriction, but we have chosen here to stick to this simpler situation.

As mentioned above, in the general case, the matching is not unitary and thus we should deal with (empty, finite or infinite) sets of substitutions. We consider a substitution application at the meta-level of the calculus represented by the operator "$_\langle\!\langle _ \rangle\!\rangle$" whose behavior is described by the meta-rule *Propagate*:

$$Propagate \quad r\langle\!\langle\{\sigma_1, \dots, \sigma_n, \dots\}\rangle\!\rangle \quad \rightsquigarrow \quad \{\sigma_1 r, \dots, \sigma_n r, \dots\}$$

Notice that since this rule operates at the meta-level of the calculus, it is different from the evaluation rules like $Fire$ and its arrow is denoted differently. A version of the calculus can also be given using explicit substitution [8].

The result of the application of a set of substitutions $\{\sigma_1, \dots, \sigma_n, \dots\}$ to a term r is the set of terms $\sigma_i r$, where $\sigma_i r$ represents the result of the (meta-)application of the substitution σ_i to the term r as detailed in Definition 2.6. Notice that when n is 0, *i.e.* the set of substitutions is empty, the resulting set of instantiated terms is also empty.

The evaluation rules of the ρ_T-calculus describe the application of a ρ-term on another one and specify the behavior of the different operators of the calculus when some arguments are sets. Following their specifications they are described in Figure 2 to 5.

2.4.1 Applying rewrite rules

The application of a rewrite rule at the root position of a term is accomplished by matching the left-hand side of the rewrite rule on the term and returning the appropriately instantiated right-hand side. It is described by the evaluation rule *Fire* in Figure 2. The rule *Fire*, like all the evaluation rules of the calculus, can be applied at any position of a ρ-term.

$$Fire \quad [l \to r](t) \quad \Longrightarrow \quad r\langle\!\langle \mathcal{S}olution(l \ll^?_T t)\rangle\!\rangle$$

FIG. 2. The evaluation rule *Fire* of the ρ_T-calculus

The central idea is that applying a rewrite rule $l \to r$ at the root (also called top) occurrence of a term t, written as $[l \to r](t)$, consists in replacing the term r by $r\langle\!\langle\Sigma\rangle\!\rangle$ where Σ is the set of substitutions obtained by T-matching l on t (*i.e.* $\mathcal{S}olution(l \ll^?_T t)$). Therefore, when the matching yields a failure represented by an empty set of substitutions, the result of the application of the rule *Propagate* and thus of the rule *Fire* is the empty set.

One can notice that the rule *Fire* can be expressed without using the meta-rule *Propagate*:

$$\begin{array}{lll} Fire & [l \to r](t) \quad \rightsquigarrow & \{\sigma_1 r, \ldots, \sigma_n r, \ldots\} \\ & & \text{where } \{\sigma_1, \ldots, \sigma_n, \ldots\} = \mathcal{S}olution(l \ll^?_T t) \end{array}$$

but we preferred the previous version for a smoother transition to the explicit version of the calculus.

We should point out that, as in λ-calculus, an application can always be evaluated. But, unlike in λ-calculus, the set of results can be empty. More generally, when matching modulo a theory T, the set of resulting matches may be empty, a singleton (as in the empty theory), a finite set (as for associativity-commutativity) or infinite (see [20]). We have thus chosen to represent the result of a rewrite rule application to a term as a set. An empty set means that the rewrite rule $l \to r$ fails to apply to t in the sense of a matching failure between l and t.

We denote by $\longrightarrow_{Fire}$ the relation induced by the evaluation rule *Fire*.

Example 2.11 Some examples of the application of the evaluation rule *Fire* are:

- $[a \to b](a) \longrightarrow_{Fire} \{b\}$
- $g(x, [x \to c](a)) \longrightarrow_{Fire} g(x, \{c\})$
- $[a \to b](c) \longrightarrow_{Fire} \emptyset$

2.4.2 Applying operators

In order to push rewrite rule application deeper into terms, we introduce the two *Congruence* evaluation rules of Figure 3. They deal with the application of a term of the form $f(u_1, \ldots, u_n)$ (where $f \in \mathcal{F}_n$) to another term of a similar form. When we

have the same head symbol for the two terms of the application $[u](v)$ the arguments of the term u are applied on those of the term v argument-wise. If the head symbols are not the same, an empty set is obtained.

$$
\begin{array}{llll}
\mathit{Cong} & [f(u_1,\ldots,u_n)](f(v_1,\ldots,v_n)) & \Longrightarrow & \{f([u_1](v_1),\ldots,[u_n](v_n))\} \\
\mathit{CongFail} & [f(u_1,\ldots,u_n)](g(v_1,\ldots,v_m)) & \Longrightarrow & \emptyset
\end{array}
$$

FIG. 3. The evaluation rules *Congruence* of the ρ_T-calculus

Remark 2.12 The *Congruence* rules are redundant with respect to the evaluation rule *Fire* modulo an appropriate transformation of the initial term. Indeed, one could notice that the application of a term $f(u_1,\ldots,u_n)$ to another ρ-term t (*i.e.* the ρ-term $[f(u_1,\ldots,u_n)](t)$) evaluates, using the rules *Cong* and *CongFail*, to the same term as the application of the ρ-term $f(x_1,\ldots,x_n) \to f([u_1](x_1),\ldots,[u_n](x_n))$ on the same term t (*i.e.* the ρ-term $[f(x_1,\ldots,x_n) \to f([u_1](x_1),\ldots,[u_n](x_n))](t)$) using the evaluation rule *Fire*. Although we can express the same computations by using only the evaluation rule *Fire*, we prefer to keep the evaluation rules *Congruence* in the calculus for an explicit use of these rules and thus, a more concise representation of terms.

2.4.3 Handling sets in the ρ_T-calculus

The reductions describing the behavior of terms containing sets are described by the evaluation rules in Figure 4:

- The rules *Distrib* and *Batch* describe the interaction between the application and the set operators,
- The rules Switch_L and Switch_R describe the interaction between the abstraction and the set operators,
- The rule *OpOnSet* describe the interaction between the symbols of the signature and the set operators.
- The rule describing the interaction between set operators will be described in the next section.

The set representation for the results of the rewrite rule application has important consequences concerning the behavior of the calculus. We can notice, in particular, that the number of set symbols is unchanged by the evaluation rules *Distrib*, *Batch*, Switch_L, Switch_R and *OpOnSet*. This way, for a derivation involving only terms that do not contain empty sets, the number of set symbols in a term counts the number of rules *Fire* and *Congruence* that have been applied for its evaluation.

The application of the set of rewrite rules $\{a \to b, a \to c\}$ to the term a (*i.e.* the ρ-term $[\{a \to b, a \to c\}](a)$) is reduced, by using the evaluation rule *Distrib*, to the set containing the application of each rule to the term a (*i.e.* the ρ-term $\{[a \to b](a), [a \to c](a)\}$). It is in particular useful when simulating ordinary term rewriting by a *set* of rewrite rules. Moreover, we can factor a set of rewrite rules

Distrib	$[\{u_1,\dots,u_n\}](v)$	$\Longrightarrow$	$\{[u_1](v),\dots,[u_n](v)\}$
Batch	$[v](\{u_1,\dots,u_n\})$	$\Longrightarrow$	$\{[v](u_1),\dots,[v](u_n)\}$
Switch$_L$	$\{u_1,\dots,u_n\}\to v$	$\Longrightarrow$	$\{u_1\to v,\dots,u_n\to v\}$
Switch$_R$	$u\to\{v_1,\dots,v_n\}$	$\Longrightarrow$	$\{u\to v_1,\dots,u\to v_n\}$
OpOnSet	$f(v_1,\dots,\{u_1,\dots,u_m\},\dots,v_n)$	$\Longrightarrow$	
	$\{f(v_1,\dots,u_1,\dots,v_n),\dots,f(v_1,\dots,u_m,\dots,v_n)\}$		

FIG. 4. The evaluation rules *Set* of the ρ_T-calculus

having the same left-hand side and use the ρ-term $a\to\{b,c\}$ which is reduced, by applying the evaluation rule *Switch*$_R$, to $\{a\to b, a\to c\}$. Thus, we can say that the ρ-term $[a\to\{b,c\}](a)$ describes the non-deterministic choice between the application of the rule $a\to b$ to the term a and the application of the rule $a\to c$ to the same term and this application is reduced to the set containing the results of the two applications, *i.e.* $\{\{b\},\{c\}\}$.

Let us consider the ρ-term $[f(a\to b)](f(a))$ which is reduced, by using the rules *Cong* and *Fire*, to $\{f(\{b\})\}$ and then, by using the rule *OpOnSet* to $\{\{f(b)\}\}$. The two set symbols corresponding to the two applications of the evaluation rules *Fire* and *Cong* are thus preserved by the application of the rule *OpOnSet*.

A result of the form $\{\}$ (*i.e.* $\emptyset$) represents the failure of a rule application and such failures are *strictly* propagated in ρ-terms by the *Set* rules. For instance, the ρ-term $g([a\to b](c),\{a\})$ is reduced to $g(\emptyset,\{a\})$ and then, by using the rule *OpOnSet*, to $\emptyset$. One should notice that in this case, the information on the number of *Fire* and *Congruence* rules used in the reduction of the sub-term $\{a\}$ is lost.

The rewrite relation generated by the evaluation rules *Fire*, *Congruence* and the *Set* rules is finer (*i.e.* contains more elements) than the standard one (without sets) and is obviously non-confluent. A reason for the non-confluence is the lack of a similar evaluation rule for the propagation of sets on sets.

2.4.4 Flattening sets in the ρ_T-calculus

We usually care about the set of results obtained by reducing the redexes and not about the exact trace of the reduction leading to these results. In what follows we present the way this behavior is described in the ρ-calculus.

We use the evaluation rule *Flat* in Figure 5 that flattens the sets and eliminates the (nested) set symbols. In this case, the information on the number of reduction steps is lost. Notice that this implies that failure (the empty set) is *not* strictly propagated on sets.

The same behavior can be described by two distinct evaluation rules: one that would just flatten the sets and thus preserve the number of set braces, and another

$$Flat \quad \{u_1,\ldots,\{v_1,\ldots,v_n\},\ldots,u_m\} \implies \{u_1,\ldots,v_1,\ldots,v_n,\ldots,u_m\}$$

FIG. 5. The evaluation rules *Flat* of the ρ_T-calculus

one that would eliminate the nested set symbols.

This behavior of the calculus could be summarized by stating that failure propagation by the *Set* rules is strict on all operators but sets. We will see later that *Fire* may induce non-strict propagations in some particular cases (see Example 4.4 on page 388).

The design decision to use sets for representing reduction results has another important consequence concerning the handling of sets with respect to matching. Indeed, sets are just used to store results and we do not wish to make them part of the theory. We are thus assuming that the matching operation used in the *Fire* evaluation rule is *not* performed modulo the set axioms. As a consequence, this requires in some cases to use a strategy that pushes set braces outside the terms whenever possible.

Every time a ρ-term is reduced using the rules *Fire* and *Congruence* of the ρ_T-calculus, a set is generated. These evaluation rules are the ones that describe the application of a rewrite rule at the top level or deeper in a term. The set obtained when applying one of the above evaluation rules can trigger the application of the other evaluation rules of the calculus. These evaluation rules deal with the (propagation of) sets and compute a "set-normal form" for the ρ-terms by pushing out the set braces and flattening the sets.

Therefore, we consider that the evaluation rules of the ρ_T-calculus consist of a set of *deduction* rules (*Fire*, *Cong*, *CongFail*) and a set of *computation* rules (*Distrib*, *Batch*, $Switch_L$, $Switch_R$, *OpOnSet*, *Flat*) and that the reduction behaves as in deduction modulo [14]. This means that we can consider the computation rules as describing a congruence modulo which the deduction rules are applied. In such an approach we say that $[f(a \to b)](f(a))$ reduces to $\{f(\{b\})\}$ which is equivalent to $\{f(b)\}$.

2.4.5 Using the ρ_T-calculus

The aim of this section is to make concrete the concepts we have just introduced by giving a few examples of ρ-terms and ρ-reductions. Many other examples could be found on the ELAN web page [51].

The ρ_T-calculus using syntactic matching (*i.e.* an empty matching theory) is denoted $\rho_\emptyset$-calculus or simply ρ-calculus when there is no ambiguity. We denote by ρ_C-calculus, ρ_A-calculus and ρ_{AC}-calculus the ρ_T-calculus with a matching theory commutative, associative and associative-commutative respectively.

Simple functional programming Let us start with the functional part of the calculus and give the ρ-terms representing some λ-terms. For example, the λ-abstraction $\lambda x.(y\ x)$, where y is a variable, is represented as the ρ-rule $x \to [y](x)$. The application of the above term to a constant a, $(\lambda x.(y\ x)\ a)$ is represented in the ρ-calculus by the application $[x \to [y](x)](a)$. This application reduces, in the λ-calculus, to the

term $(y\ a)$ while in the ρ-calculus the result of the reduction is the singleton $\{[y](a)\}$. When a functional representation $f(x)$ is chosen, the λ-term $\lambda x.f(x)$ is represented by the ρ-term $x \to f(x)$ and a similar result is obtained for its application. One should notice that for ρ-terms of this form (*i.e.* that have a variable as a left-hand side) the syntactic matching performed in the ρ-calculus is trivial, *i.e.* it never fails and gives only one result.

There is no difficulty to represent more elaborate λ-terms in the ρ-calculus. Let us consider the term $\lambda x.f(x)\ (\lambda y.y\ a)$ with the following β-derivation: $\lambda x.f(x)\ (\lambda y.y\ a) \longrightarrow_\beta \lambda x.f(x)\ a \longrightarrow_\beta f(a)$. The same derivation can be recovered in the ρ-calculus for the corresponding ρ-term: $[x \to f(x)]([y \to y](a)) \longrightarrow_{Fire} [x \to f(x)](\{a\}) \longrightarrow_{Batch} \{[x \to f(x)](a)\} \longrightarrow_{Fire} \{\{f(a)\}\} \longrightarrow_{Flat} \{f(a)\}$. Of course, several reduction strategies can be used in the λ-calculus and reproduced accordingly in the ρ-calculus. Indeed, we will see in Section 3.1 that the ρ-calculus strictly embeds the λ-calculus.

Rewriting Now, if we introduce contextual information in the left-hand sides of the ρ-rules we obtain classical rewrite rules as $f(a) \to f(b)$ or $f(x) \to g(x,x)$. When we apply such a rewrite rule, the matching can fail and consequently, the application of the rewrite rule can fail. As we have already insisted in the previous sections, the failure of a rewrite rule is not a meta-property in the ρ-calculus but is represented by an empty set (of results). For example, in standard term rewriting we say that the application of the rule $f(a) \to f(b)$ to the term $f(c)$ fails and therefore the term is unchanged. On the contrary, in the ρ-calculus the corresponding term $[f(a) \to f(b)](f(c))$ evaluates to $\emptyset$.

Since, in the ρ-calculus, there is no restriction on the rewrite rules construction, a rewrite rule may use a variable as left-hand side, as in $x \to x+1$, or it may introduce new variables, as in $f(x) \to g(x,y)$. The free variables of the rewrite rules from the ρ-calculus allow us to dynamically build classical rewrite rules. For example, in the application $[y \to (f(x) \to g(x,y))](a)$, the variable y is free in the rewrite rule $f(x) \to g(x,y)$ but bound in the rule $y \to (f(x) \to g(x,y))$. The above application is reduced to the set $\{f(x) \to g(x,a)\}$ containing a classical rewrite rule.

By using free variables in the right-hand side of a rewrite rule we can also "parameterize" the rules by "strategies", as in the term $y \to [f(x) \to [y](x)](f(a))$ where the term to be applied to x is not explicit in the rule $f(x) \to [y](x)$. When reducing the application $[y \to [f(x) \to [y](x)](f(a))](a \to b)$, the variable y from the rewrite rule is instantiated to $a \to b$ and thus, the result of the reduction is $\{b\}$.

Non-determinism When the matching is done modulo an equational theory we obtain interesting behaviors.

An associative matching theory allows us, for example, to express the fact that an expression can be parenthesized in different ways. Take, for example, the list operator $\circ$ that appends two lists with elements of a given sort *Elem*. Any object of sort *Elem* represents a list consisting of this only object. If we define the operator $\circ$ as associative, the rewrite rule describing the decomposition of a list can be written in the associative ρ_A-calculus $l \circ l' \to l$. When applying this rule to the list $a \circ b \circ c \circ d$ we obtain as result the ρ-term $\{a, a \circ b, a \circ b \circ c\}$. If the operator $\circ$ had not been defined as associative, we would have obtained as the result of the same rule application one of the singletons $\{a\}$ or $\{a \circ b\}$ or $\{a \circ (b \circ c)\}$ or $\{(a \circ b) \circ c\}$, depending on the way the term $a \circ b \circ c \circ d$ is parenthesized.

A commutative matching theory allows us, for example, to express the fact that the order of the arguments is not significant. Let us consider a commutative operator $\oplus$ and the rewrite rule $x \oplus y \to x$ that selects one of the elements of the tuple $x \oplus y$. In the commutative ρ_C-calculus, the application $[x \oplus y \to x](a \oplus b)$ evaluates to the set $\{a, b\}$ that represents the set of non-deterministic choices between the two results. In standard rewriting, the result is not well defined; should it be a or b?

We can also use an associative-commutative theory like, for example, when an operator describes multi-set formation. Let us go back to the $\circ$ operator, but this time we define it as associative-commutative and we use the rewrite rule $x \circ x \circ L \to L$ that eliminates doubleton from lists of sort $Elem$. Since the matching is done modulo associativity-commutativity, this rule eliminates the doubleton no matter what is their position in the structure built using the $\circ$ operator. For instance, in the ρ_{AC}-calculus the application $[x \circ x \circ L \to L](a \circ b \circ c \circ a \circ d)$ evaluates to $\{b \circ c \circ d\}$: the search for the two equal elements is done thanks to associativity and commutativity.

Another facility is due to the use of sets for handling non-determinism. This allows us to easily express the non-deterministic application of a set of rewrite rules to a term. Let us consider, for example, the operator $\otimes$ as a syntactic operator. If we want the same behavior as before for the selection of each element of the couple $x \otimes y$, two rewrite rules should be non-deterministically applied as in the following reduction: $[\{x \otimes y \to x, x \otimes y \to y\}](a \otimes b) \longrightarrow_{Distrib} \{[x \otimes y \to x](a \otimes b), [x \otimes y \to y](a \otimes b)\}$ $\longrightarrow_{Fire} \{\{a\}, \{b\}\} \longrightarrow_{Flat} \{a, b\}$.

2.5 Evaluation strategies for the ρ_T-calculus

The last component of a calculus, *i.e.* the strategy $\mathcal{S}$ guiding the application of its evaluation rules, is crucial for obtaining good properties for the ρ-calculus. For example, the main property analyzed for the ρ-calculus is confluence and we will see that if the rule $Fire$ is applied under no conditions at any position of a ρ-term, confluence does not hold.

Let us now define formally the notion of strategy. We specialize here to the ρ-calculus, and the general definition can be found in [35].

Definition 2.13 An *evaluation strategy* in the ρ-calculus is a subset of the set of all possible derivations.

For example, the $\mathcal{ALL}$ strategy is the set of *all* derivations, *i.e.* it imposes no restrictions. The empty strategy does not allow any reduction. Standard strategies are call by value or by name, leftmost innermost or outermost, lazy, needed.

The reasons for the non-confluence of the calculus are explained in Section 4 and a solution is proposed for obtaining a confluent calculus. The confluent strategy can be given explicitly or as a condition on the application of the rule $Fire$.

2.6 Summary

Starting from the notions introduced in the previous sections we give the definition of the ρ_T-calculus.

Definition 2.14 Given a set $\mathcal{F}$ of function symbols, a set $\mathcal{X}$ of variables, a theory

T on $\varrho(\mathcal{F}, \mathcal{X})$ terms having a decidable matching problem, we call ρ_T-calculus (or generically rewriting calculus) a calculus defined by:

1. a non-empty subset $\varrho_{-}(\mathcal{F}, \mathcal{X})$ of the $\varrho(\mathcal{F}, \mathcal{X})$ terms,
2. the (higher-order) substitution application to terms as defined in Section 2.2,
3. the theory T,
4. the set of evaluation rules $\mathcal{E}$: *Fire*, *Cong*, *CongFail*, *Distrib*, *Batch*, $Switch_L$, $Switch_R$, *OpOnSet*, *Flat*,
5. an evaluation strategy $\mathcal{S}$ that controls the application of the evaluation rules. The set $\varrho_{-}(\mathcal{F}, \mathcal{X})$ should be stable under the strategy controlled application of the evaluation rules.

We use the notation $\rho_T = (\varrho_{-}(\mathcal{F}, \mathcal{X}), T, \mathcal{S})$ to make apparent the main components of the rewriting calculus under consideration.

When the parameters of the general calculus are replaced with some specific values, different variants of the calculus are obtained. The remainder of this paper will be devoted, mainly, to the study of a specific instance of the ρ_T-calculus: the ρ-calculus.

2.7 Definition of the ρ-calculus

We define the ρ-calculus as the ρ_T-calculus where the matching theory T is restricted to first-order syntactic matching. As an instance of Definition 2.14 we get:

Definition 2.15 The ρ-calculus is the calculus defined by:

- the subset $\varrho_{\emptyset}(\mathcal{F}, \mathcal{X})$ of $\varrho(\mathcal{F}, \mathcal{X})$ whose rewrite rules are restricted to be of the form $u \to v$ where $u \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, *i.e.* u is a first-order term and thus does not contain any set, application or abstraction symbol,
- the higher-order substitution application to terms,
- the matching theory $T = \emptyset$, *i.e.* first-order syntactic matching,
- the set of evaluation rules $\mathcal{R}$ presented in Figure 6 (*i.e.* all the rules of the ρ-calculus but $Switch_L$),
- the evaluation strategy $\mathcal{ALL}$ that imposes no conditions on the application of the evaluation rules.

The ρ-calculus is therefore defined as the calculus $\rho_{\emptyset} = (\varrho_{\emptyset}(\mathcal{F}, \mathcal{X}), \emptyset, \mathcal{ALL})$.

Example 2.16 With the exception of the last term, all the ρ-terms from Example 2.3 are $\rho_{\emptyset}$-terms.

The following remarks should be made with respect to the restrictions introduced in the ρ-calculus:

- Since first-order syntactic matching is unitary (*i.e.* the match, when it exists, is unique) the meta-rule *Propagate* from Section 2.4 gives always as result either the singleton $\{\sigma r\}$ or the empty set. Hence, the evaluation rule *Fire* can be replaced by the following simpler two rules:

$$\begin{array}{lll} Fire' & [l \to r](\sigma l) & \Longrightarrow \quad \{\sigma r\} \\ Fire'' & [l \to r](t) & \Longrightarrow \quad \emptyset \\ & & \qquad \text{if there exists no } \sigma \text{ s.t. } \sigma l = t \end{array}$$

Fire	$[l \to r](t)$	$\Longrightarrow$	$\{\sigma r\}$ where $\{\sigma\} = \mathcal{S}olution(l \ll^?_T t)$
Cong	$[f(u_1,\ldots,u_n)](f(v_1,\ldots,v_n))$	$\Longrightarrow$	$\{f([u_1](v_1),\ldots,[u_n](v_n))\}$
CongFail	$[f(u_1,\ldots,u_n)](g(v_1,\ldots,v_m))$	$\Longrightarrow$	$\emptyset$
Distrib	$[\{u_1,\ldots,u_n\}](v)$	$\Longrightarrow$	$\{[u_1](v),\ldots,[u_n](v)\}$
Batch	$[v](\{u_1,\ldots,u_n\})$	$\Longrightarrow$	$\{[v](u_1),\ldots,[v](u_n)\}$
$Switch_R$	$u \to \{v_1,\ldots,v_n\}$	$\Longrightarrow$	$\{u \to v_1,\ldots,u \to v_n\}$
OpOnSet	$f(v_1,\ldots,\{u_1,\ldots,u_m\},\ldots,v_n)$	$\Longrightarrow$	$\{f(v_1,\ldots,u_1,\ldots,v_n),\ldots,f(v_1,\ldots,u_m,\ldots,v_n)\}$
Flat	$\{u_1,\ldots,\{v_1,\ldots,v_n\},\ldots,u_m\}$	$\Longrightarrow$	$\{u_1,\ldots,v_1,\ldots,v_n,\ldots,u_m\}$

FIG. 6. The evaluation rules of the ρ-calculus

- The evaluation rule $Switch_L$ can never be used in the ρ-calculus due to the restricted syntax imposed on $\rho_\emptyset$-terms.
- For a specific instance of the ρ_T-calculus, there is a strong relationship between the terms allowed on the left-hand side of the rule and the theory T. Intuitively, the theory T should be powerful enough to fire rule applications in a way consistent with the intended rewriting. For instance, it seems more interesting to use higher-order matching instead of syntactic or equational matching when the left-hand sides of rules contain abstractions and applications. This explains the restriction imposed in the ρ-calculus for the formation of left-hand sides of rules.
- The term restrictions are made only on the left-hand sides of rewrite rules and not on the right-hand side and this clearly leads to more terms than in λ-calculus or in term rewriting.
- The ρ-calculus is not terminating as $[\omega](\omega)$ is a ρ-term (see Example 2.3).

The case of decidable finitary equational theories will induce more technicalities but is conceptually similar to the case of the empty theory. The case of theories with infinitary or undecidable matching problems could be treated using constraint ρ-terms in the spirit of [34], and will be studied in forthcoming works.

3 Encoding λ-calculus and term rewriting in the ρ-calculus

The aim of this section is to show in detail how the ρ-calculus can be used to give a natural encoding of the λ-calculus and term rewriting.

3.1 Encoding the λ-calculus

We briefly present some of the notions used in the λ-calculus, such as β-redex and β-reduction, that will be used in this part of the paper. The reader should refer to [25] and [2] for a detailed presentation.

Let $\mathcal{X}$ be a set of variables, written x, y, etc. The terms of the λ-calculus are inductively defined by:

$$a ::= x \mid (a\ a) \mid \lambda x.a$$

Definition 3.1 The β-reduction is defined by the rule:

$$Beta \quad (\lambda x.M\ N) \quad \rightsquigarrow \quad \{x/N\}M$$

Any term of the form $(\lambda x.M)N$ is called a *β-redex*, and the term $\{x/N\}M$ is traditionally called its *contractum*. If a term P contains a redex, P can be *β-contracted* into P' which is denoted:

$$P \ \longrightarrow_{\beta} \ P'.$$

If Q is obtained from P by a finite (possibly empty) number of β-contractions we say that P *β-reduces* to Q and we denote:

$$P \ \stackrel{*}{\longrightarrow}_{\beta} \ Q.$$

Let us consider a restriction of the set of ρ-terms, denoted $\mathcal{F}_\lambda$, and inductively defined as follows:

$$\rho_\lambda\text{-terms} \quad t \quad ::= \quad x \mid \{t\} \mid [t](t) \mid x \rightarrow t$$

where $x \in \mathcal{X}$.

Definition 3.2 The ρ_λ-calculus is the ρ-calculus defined by:

- the $\mathcal{F}_\lambda$ terms,
- the higher-order substitution application to terms,
- the (matching) theory $T = \emptyset$,
- the set of evaluation rules of the ρ-calculus,
- the evaluation strategy $\mathcal{ALL}$ that imposes no conditions on the application of the evaluation rules.

Compared to the syntax of the general ρ-calculus, the rewrite rules allowed in the ρ_λ-calculus can only have a variable as left-hand side. Additionally, all the sets are singletons, hence one could consider an encoding not using sets. For uniformity purposes, we chose to stick to the same encoding approach.

Because of the syntactic restrictions we have just imposed, the evaluation rules of the ρ-calculus specialize to the ones described in Figure 7.

The evaluation rule $Fire_\lambda$ initiates in the ρ-calculus (as the β-rule in the λ-calculus) the application of a substitution to a term. The rules *Congruence* are not used and the rules *Set* and *Flat* can be specialized to singletons and describe how to push out the set braces.

$$\begin{array}{llcl}
Fire_\lambda & [x \to r](t) & \Longrightarrow & \{\{x/t\}r\ \} \\
Distrib_\lambda & [\{u\}](v) & \Longrightarrow & \{[u](v)\} \\
Batch_\lambda & [v](\{u\}) & \Longrightarrow & \{[v](u)\} \\
Switch_\lambda & x \to \{v\} & \Longrightarrow & \{x \to v\} \\
Flat_\lambda & \{\{v\}\} & \Longrightarrow & \{v\}
\end{array}$$

FIG. 7. The evaluation rules of the ρ_λ-calculus

An immediate consequence of the restricted syntax of the ρ_λ-calculus is that the matching performed in the evaluation rule $Fire_\lambda$ always succeeds and the solution of the matching equation that is necessarily of the form $x \ll^?_\emptyset t$ is always the singleton $\{\{x/t\}\}$.

At this moment we can notice that any λ-term can be represented by a ρ-term. The function φ that transforms terms in the syntax of the λ-calculus into the syntax of the ρ_λ-calculus is defined by the following transformation rules:

$$\begin{array}{lcl}
\varphi(x) & = & x, \text{ if } x \text{ is a variable} \\
\varphi(\lambda x.t) & = & x \to \varphi(t) \\
\varphi(t\ u) & = & [\varphi(t)](\varphi(u))
\end{array}$$

A similar translation function can be used in order to transform terms in the syntax of the ρ_λ-calculus into the syntax of the λ-calculus:

$$\begin{array}{lcl}
\delta(x) & = & x, \text{ if } x \text{ is a variable} \\
\delta(\{t\}) & = & \delta(t) \\
\delta([t](u)) & = & (\delta(t)\ \delta(u)) \\
\delta(x \to t) & = & \lambda x.\delta(t)
\end{array}$$

The reductions in the λ-calculus and in the ρ_λ-calculus are equivalent modulo the notations for the application and the abstraction and the handling of sets:

Proposition 3.3 Given two λ-terms t and t', if $t \longrightarrow_\beta t'$ then $\varphi(t) \xrightarrow{*}_{\rho_\lambda} \{\varphi(t')\}$. Given two ρ_λ-terms u and u', if $u \longrightarrow_{\rho_\lambda} u'$ then $\delta(u) \xrightarrow{*}_\beta \delta(u')$.

PROOF. We use an induction on $\longrightarrow_\beta$ and $\longrightarrow_{\rho_\lambda}$ respectively:

- If t is a variable x, then $t' = x$ and $\varphi(t) = \varphi(t') = x$.
- If $t = \lambda x.u$ then $t' = \lambda x.u'$ with $u \longrightarrow_\beta u'$ and we have $\varphi(t) = x \to \varphi(u)$. By induction, we have $\varphi(u) \xrightarrow{*}_{\rho_\lambda} \{\varphi(u')\}$, and thus

$$\varphi(t) = x \to \varphi(u) \quad \xrightarrow{*}_{\rho_\lambda} \quad x \to \{\varphi(u')\} \quad \longrightarrow_{Switch_\lambda} \quad \{x \to \varphi(u')\} = \{\varphi(t')\}$$

- If $t = (u\ v)$ then we have either $t' = (u'\ v)$ with $u \longrightarrow_\beta u'$, or $t' = (u\ v')$ with $v \longrightarrow_\beta v'$, or $t = \lambda x.u\ v$ and $t' = \{x/v\}u$.
  In the first case, we apply induction and we obtain

$$\varphi(t) = [\varphi(u)](\varphi(v)) \xrightarrow{*}_{\rho_\lambda} [\{\varphi(u')\}](\varphi(v)) \longrightarrow_{Distrib_\lambda} \{[\varphi(u')](\varphi(v))\} = \{\varphi(t')\}.$$

The second case is similar,

$$\varphi(t) = [\varphi(u)](\varphi(v)) \xrightarrow{*}_{\rho_\lambda} [\{\varphi(u)\}](\varphi(v')) \longrightarrow_{Distrib_\lambda} \{[\varphi(u)](\varphi(v'))\} = \{\varphi(t')\}.$$

In the third case $\varphi(t) = [x \to \varphi(u)](\varphi(v))$ and

$$\varphi(t) = [x \to \varphi(u)](\varphi(v)) \longrightarrow_{Fire_\lambda} \{\{x/\varphi(v)\}\varphi(u)\} = \varphi(\{x/v\}u) = \varphi(t').$$

Since the application of a substitution is the same in the λ-calculus and the ρ-calculus, we have, due to the definition of φ, $\varphi(\{x/v\}u) = \{x/\varphi(v)\}\varphi(u)$ and thus, the property is verified.

Since in the ρ_λ-calculus we can have only singletons and the δ transformation strips off the set symbols, the application of the evaluation rules $Distrib_\lambda$, $Batch_\lambda$, $Switch_\lambda$ and $Flat_\lambda$ corresponds to the identity in the λ-calculus.

- If $t = [\{u\}](v)$ then we have $t \longrightarrow_{Distrib_\lambda} \{[u](v)\}$. Since $\delta([\{u\}](v)) = \delta(u)\ \delta(v)$ and $\delta(\{[u](v)\}) = \delta(u)\ \delta(v)$, the property is verified.
- If $t = [x \to u](v)$ then $t \longrightarrow_{Fire_\lambda} \{\{x/v\}u\}$. We have

$$\delta(t) = \lambda x.\delta(u)\ \delta(v) \longrightarrow_\beta \{x/\delta(v)\}\delta(u)\} = \delta(\{x/v\}u) = \delta(t').$$

The other cases are very similar to the first one and to their correspondents from the first part.

□

Example 3.4 We consider the three combinators $I = \lambda x.x$, $K = \lambda xy.x$ and $S = \lambda xyz.xz(yz)$ and their representation in the ρ-calculus:

- $I = x \to x$,
- $K = x \to (y \to x)$,
- $S = x \to (y \to (z \to [[x](z)]([y](z))))$.

and, as expected, to a reduction $SKK \xrightarrow{*}_\beta I$ in the λ-calculus it corresponds the ρ_λ-reduction $[[S](K)](K) \xrightarrow{*}_{\rho_\lambda} \{I\}$.

$$\begin{aligned}
[[S](K)](K) = {} & [[x \to (y \to (z \to [[x](z)]([y](z))))](x \to (y \to x))](x \to (y \to x)) \longrightarrow_{\rho_\lambda} \\
& [\{y \to (z \to [[x \to (y \to x)](z)]([y](z)))\}](x \to (y \to x)) \longrightarrow_{\rho_\lambda} \\
& \{[y \to (z \to [[x \to (y \to x)](z)]([y](z)))](x \to (y \to x))\} \longrightarrow_{\rho_\lambda} \\
& \{[y \to (z \to [\{y \to z\}]([y](z)))](x \to (y \to x))\} \longrightarrow_{\rho_\lambda} \\
& \{\{[y \to (z \to [y \to z]([y](z)))](x \to (y \to x))\}\} \longrightarrow_{\rho_\lambda} \\
& \{\{[y \to (z \to \{z\})](x \to (y \to x))\}\} \longrightarrow_{\rho_\lambda} \\
& \{\{\{[y \to (z \to z)](x \to (y \to x))\}\}\} \longrightarrow_{\rho_\lambda} \\
& \{\{\{\{z \to z\}\}\}\} \longrightarrow_{\rho_\lambda} \\
& \{z \to z\} = \{I\}
\end{aligned}$$

The need for adding a set symbol comes from the fact that in the ρ-calculus we are mainly interested in the application of terms to some other terms. From this point of view, the application of a term t to another term u reduces to the same thing as the application of the term $\{t\}$ to the same term u.

In the ρ_λ-calculus, we could have introduced an evaluation rule eliminating all set symbols. But as soon as failure, represented by the empty set, and non-determinism, represented by sets with more than one element, are introduced such an evaluation rule will not be meaningful anymore.

The confluence of the λ-calculus holds for any complete reduction strategy (*i.e.* a strategy that does not leave any redex un-reduced) and we would expect the same result for its ρ-representation. As we have already noticed, since in the ρ_λ-calculus all the rewrite rules are left-linear and all the sets are singletons, the confluence conditions that will be presented in Section 4.2 are always satisfied. Therefore, the evaluation rule $Fire_\lambda$ can be used on any ρ_λ-application without losing the confluence of the ρ_λ-calculus.

Proposition 3.5 The ρ_λ-calculus is confluent.

Notice finally that using the same technique, the λ-calculus with patterns of [50] can be encoded as a sub-calculus of the ρ-calculus.

3.2 Encoding finite rewrite sequences

As far as it concerns term rewriting, we just recall the basic notions that are consistent with [16, 4] to which the reader is referred for a more detailed presentation.

A *rewrite theory* is a 4-tuple $\mathcal{R} = (\mathcal{X}, \mathcal{F}, E, R)$ where $\mathcal{X}$ is a given countably infinite set of variables, $\mathcal{F}$ a set of ranked function symbols, E a set of $\mathcal{T}(\mathcal{F}, \mathcal{X})$-equalities, and R a set of rewrite rules of the form $l \to r$ where $l, r \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ satisfying $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$.

In what follows we consider $E = \emptyset$ but we conjecture that all the results concerning the encoding of rewriting in ρ-calculus can be smoothly extended to any equational theory E.

Since the rewrite rules are trivially ρ-terms, the representation of rewrite sequences in the ρ-calculus is quite simple. We consider a restriction of the ρ-calculus where the right-hand sides of rewrite rules are terms of $\mathcal{T}(\mathcal{F}, \mathcal{X})$. The rewrite rules are trivially translated in the ρ-calculus and the application of a rewrite rule at the top position of a term is represented using the ρ-operator $[_](_)$.

We want to show that for any derivation in a rewriting theory, a corresponding reduction can be found in the ρ-calculus. If we consider that a sub-term w of a term t is reduced to w' by applying some rewrite rule $(l \to r)$ and thus,

$$t_{\lceil w \rceil_p} \longrightarrow_{\mathcal{R}} t_{\lceil w' \rceil_p}$$

then, we can build immediately the ρ-term $t_{\lceil [l \to r](w) \rceil_p}$ with the reduction:

$$t_{\lceil [l \to r](w) \rceil_p} \longrightarrow_\rho t_{\lceil \{w'\} \rceil_p} \xrightarrow{*}_\rho \{t_{\lceil w' \rceil_p}\}.$$

The above construction method for the ρ-term with a ρ-reduction similar to that of the term t according to the rule $l \to r$ is very easy but allows us to find the correspondence for only one rewrite step. It is not easy to extend this representation for an unspecified number of reduction steps w.r.t. a set of rewrite rules and a systematic method for the construction of the corresponding ρ-term is desirable.

Proposition 3.6 Given a rewriting theory $\mathcal{T}_\mathcal{R}$ and two first order ground terms $t, t' \in \mathcal{T}(\mathcal{F})$ such that $t \xrightarrow{*}_\mathcal{R} t'$. Then, there exist the ρ-terms $u_1, \ldots, u_n$ built using the rewrite rules in $\mathcal{R}$ and the intermediate steps in the derivation $t \xrightarrow{*}_\mathcal{R} t'$ such that we have $[u_n](\ldots[u_1](t)\ldots) \xrightarrow{*}_{\rho_\emptyset} \{t'\}$.

PROOF. We use induction on the length of the derivation $t \xrightarrow{*}_\mathcal{R} t'$.

The base case: $t \xrightarrow{0}_\mathcal{R} t$ (derivation in 0 steps)

We have immediately $[x \to x](t) \xrightarrow{0}_{\rho_\emptyset} \{t\}$.

Induction: $t \xrightarrow{n}_\mathcal{R} t'$ (derivation in n steps)

We consider that the rewrite rule $l \to r$ is applied at position p of the term $t'{}_{\lceil w \rceil_p}$ obtained after $n-1$ reduction steps,

$$t \quad \xrightarrow{n-1}_\mathcal{R} \quad t'{}_{\lceil w \rceil_p} \quad \longrightarrow_{l \to r, p} \quad t'{}_{\lceil \theta r \rceil_p}$$

where θ is the grafting such that $\theta l = w$.

By induction, there exist the ρ-terms $u_1, \ldots, u_{n-1}$ such that we have the reduction $[u_{n-1}](\ldots[u_1](t)\ldots) \xrightarrow{*}_{\rho_\emptyset} \{t'{}_{\lceil w \rceil_p}\}$. We consider the ρ-term $u_n = t'{}_{\lceil l \to r \rceil_p}$ and we obtain the reduction

$[u_n](\ldots[u_1](t)\ldots) \quad \xrightarrow{*}_{\rho_\emptyset} \quad [t'{}_{\lceil l \to r \rceil_p}](\{t'{}_{\lceil w \rceil_p}\}) \quad \longrightarrow_{Batch} \quad \{[t'{}_{\lceil l \to r \rceil_p}](t'{}_{\lceil w \rceil_p})\}$
$\xrightarrow{*}_{Congruence} \quad \{\{t'{}_{\lceil [l \to r](w) \rceil_p}\}\} \quad \longrightarrow_{Fire} \quad \{\{t'{}_{\lceil \{\theta' r\} \rceil_p}\}\} \quad \xrightarrow{*}_{OpOnSet} \quad \{\{\{t'{}_{\lceil \theta' r \rceil_p}\}\}\}$
$\xrightarrow{*}_{Flat} \quad \{t'{}_{\lceil \theta' r \rceil_p}\}$

where the substitution θ' is such that $\{\theta'\} = \mathcal{S}olution(l \ll^?_\emptyset w)$.

Since $\theta = \theta'$ and in this case substitution and grafting are identical, we obtain $t'{}_{\lceil \theta' r \rceil_p} = t'{}_{\lceil \theta r \rceil_p}$.

□

Until now we have used the evaluation rule *Cong* for constructing the reduction

$$[t^n{}_{\lceil l_n \to r_n \rceil_{p_n}}](\ldots[t^2{}_{\lceil l_2 \to r_2 \rceil_{p_2}}]([t^1{}_{\lceil l_1 \to r_1 \rceil_{p_1}}](t))\ldots) \quad \xrightarrow{*}_\rho \quad \{t'\}$$

that corresponds, in the ρ-calculus, to the reduction, in the rewrite theory,

$$t = t^1{}_{\lceil w_1 \rceil_{p_1}} \quad \longrightarrow_{l_1 \to r_1, p_1} \quad t^2{}_{\lceil w_2 \rceil_{p_2}} \quad \longrightarrow_{l_2 \to r_2, p_2} \quad \ldots \quad \longrightarrow_{l_n \to r_n, p_n} \quad t^n{}_{\lceil w_n \rceil_{p_n}} = t'$$

As explained in Section 2.4, to any reduction performed using the rule *Cong* corresponds a reduction that is done using the rule *Fire*. Starting from the term u corresponding to a reduction in n (*Cong*) steps we build the term u' that reduces to the same term as u but using *Fire* reductions:

$$[t^n{}_{\lceil l_n \rceil_{p_n}} \to t^n{}_{\lceil r_n \rceil_{p_n}}](\ldots([t^1{}_{\lceil l_1 \rceil_{p_1}} \to t^1{}_{\lceil r_1 \rceil_{p_1}}](t))\ldots) \quad \xrightarrow{*}_\rho \quad \{t'\}$$

Remark 3.7 One can notice that the terms u_i used in the proof above are similar to the proof terms used in labeled rewriting logic [39]. Indeed we can see the ρ-terms as a generalization of such proof terms where the ";" is used as a notation for the composition of terms, *i.e.* $[u]([v](t))$ is denoted $[v; u](t)$.

4 The confluence of ρ-calculus

It is easy to see, and we provide typical examples just below, that the ρ-calculus is non-confluent. The main reason for the confluence failure comes from the introduction in

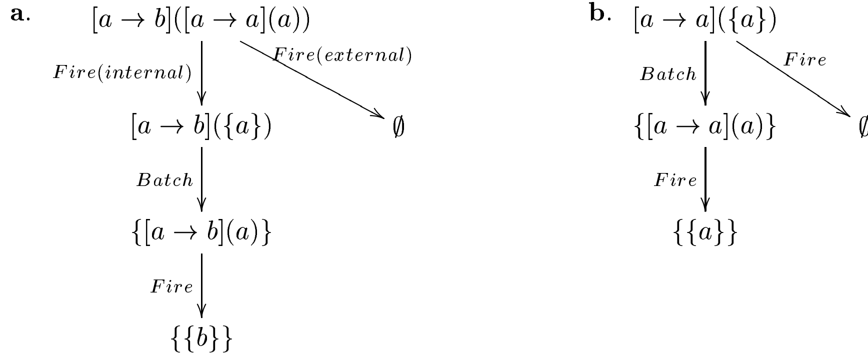
the syntax of the new function symbols for denoting sets, abstraction and application. It results in a conflict between the use of syntactic matching and the set representation for the reductions results. This leads, on one hand, to undesirable matching failures due to terms that are not completely evaluated or not instantiated. On the other hand, we can have sets with more than one element that can lead to undesirable results in a non-linear context or empty sets that are not strictly propagated. In this section, we summarize the results of [8] to which the reader is referred for full details. In particular we show on typical examples the confluence problems and we give a sufficient condition on the evaluation strategy of the ρ-calculus that allows to restore confluence.

4.1 The raw ρ-calculus is not confluent

Let us begin to show typical examples of confluence failure. A first such situation occurs when reducing a (sub-)term of the form $u = [l \to r](t)$ by matching l and t and when either t contains a redex, or u is redex.

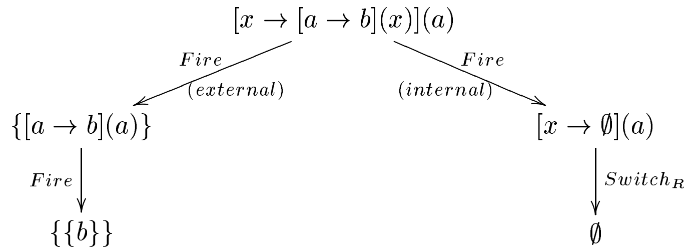
In Example 4.1.a the non-confluence is obtained when a matching failure results from a non-reduced sub-term of t but succeeds when the sub-term is reduced. A similar situation is obtained when the evaluation rule *Fire* gives the $\emptyset$ result due to a matching failure but the application of another evaluation rule before the rule *Fire* leads to a non-empty set as in Example 4.1.b.

Example 4.1



In Example 4.2 one can notice that a term can be reduced to an empty set because of a matching failure implying its bound variables. The result can be different from the empty set if the reductions of the sub-terms containing the respective variables are carried out only after the instantiation of these variables.

Example 4.2

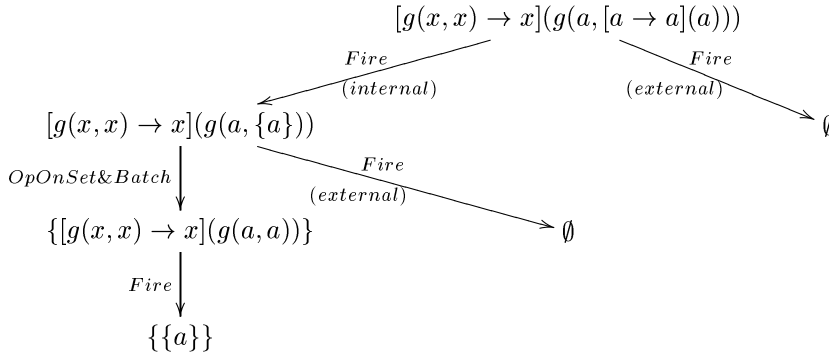


In order to avoid this kind of situation we should prevent the reduction of an application $[l \to r](t)$ if the matching between the terms l and t fails due to the matching rules $VariableClash$ (Example 4.2) or $SymbolClash$ (Example 4.1.a, 4.1.b) and either some variables are not instantiated or some of the terms are not reduced, or the term t is a set.

The matching rules $VariableClash$ and $SymbolClash$ would be never applied if the set of functional positions of the term l was a subset of the set of functional positions of the term t. This is not the case in Example 4.2 where, in the term $[a \to b](x)$, a is a functional position and the corresponding position in the argument of the rewrite rule application is the variable position x. In Example 4.1.a and Example 4.1.b a functional position in the left-hand side of the rewrite rule corresponds to an abstraction and set position respectively and thus, the condition is not satisfied.

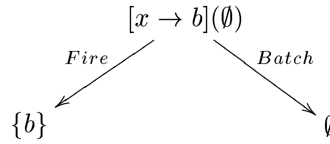
Therefore, we could consider that the evaluation rule $Fire$ is applied only when the condition on the functional positions is satisfied. Unfortunately, such a condition will not suffice for avoiding a non-appropriate matching failure due to the application of the rule $MergingClash$. As shown in Example 4.3, such a situation can be obtained if the left-hand side of the rewrite rule to be applied is not linear.

Example 4.3



Another pathological case arises when the term t contains an empty set or a subterm that can be reduced to the empty set. Indeed, the application of the rule $Fire$ can lead to the non-propagation of the failure and thus, to non-confluence as in the next example:

Example 4.4

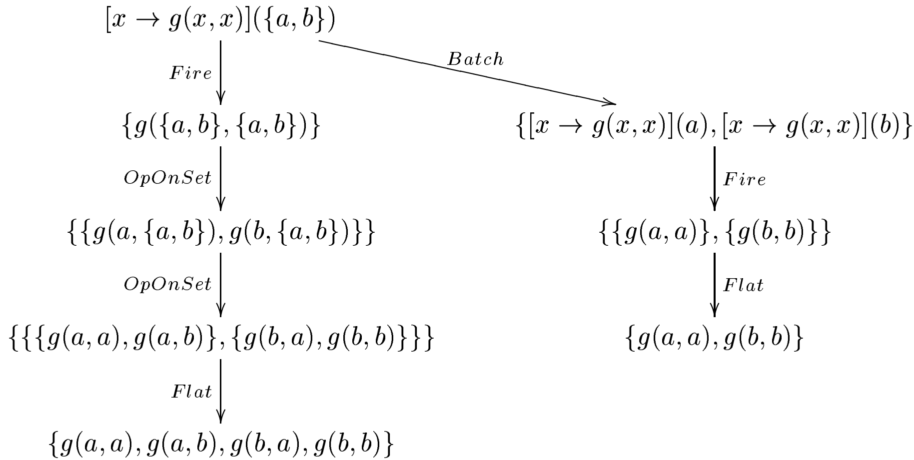


We mention that a rewrite rule is *quasi-regular* if the set of variables of the left-hand side is included in the set of variables of the right-hand side. In Section 4.2 we give a formal definition for the notion of quasi-regular rewrite rule that takes into consideration all the operators of the ρ-calculus. We have already seen in Example 4.4 that the non-propagation of the failure is obtained when non-quasi-regular rewrite rules are applied to a term containing $\emptyset$. When a quasi-regular rewrite rule is applied to a term containing $\emptyset$, the empty set is present in the term resulting from the application

of a substitution of the form $\{x/\emptyset\}$ to the right-hand side of the rewrite rule (unlike in Example 4.4) and thus, the appropriate propagation of the $\emptyset$ is guaranteed.

Another nasty situation, well known, in particular in graph rewriting, is obtained due to uncontrolled copies of terms. When applying a non-right-linear rewrite rule to a term that contains sets with more than one element, or terms that can be reduced to such sets, we obtain undesirable results as in Example 4.5.

Example 4.5



$\{g(a,a),g(a,b),g(b,a),g(b,b)\}$

To sum-up, the non-confluence is due to the application of the evaluation rule *Fire* too early in a derivation and the typical situations that we want to avoid consist in using the rule *Fire* for reducing an application:

- containing non-instantiated variables,
- containing non-reduced terms,
- containing a non-left-linear rewrite rule,
- of a non-right-linear rewrite rule to a term containing sets with more than one element,
- of a non-quasi-regular rewrite rule to a term containing empty sets.

We can notice that if we assume the computation rules (see Section 2.4) to be applied eagerly, then some, but unfortunately not all of the above confluence problems vanish. In particular, non-confluence examples involving sets, as Example 4.4 and Example 4.5, are overcome by an eager application of the computation rules.

4.2 Enforcing confluence using strategies

As we have just seen in the previous section, the possibility of having empty sets or sets with more than one element leads immediately to non-confluent reductions implying the evaluation rules *Fire* and *Congruence*. But the confluence could be restored under an appropriate evaluation strategy and, in particular, this strategy should guarantee a strict failure propagation and an appropriate handling of the sets with more than one element.

A first possible approach consists in reducing a ρ-term by initially applying all the rules handling the sets (*Distrib*, *Batch*, $Switch_L$, $Switch_R$, *OpOnSet*, *Flat*), *i.e.* the computation rules, and only when none of these rules can be applied, apply one of the rules *Fire*, *Cong*, *CongFail*, *i.e.* the deduction rules, to the terms containing no sets.

But an application can be reduced, by using the rule *Fire*, to an empty set or to a set containing several elements and thus, this strategy can still lead, as previously, to non-confluent reductions. Another disadvantage of this approach is that for no restriction of the ρ-calculus the proposed strategy is reduced to the trivial strategy $\mathcal{ALL}$.

Since the sets (empty or having more than one element) are the main cause of the non-confluence of the calculus, a natural strategy consists in reducing the application of a rewrite rule by respecting the following steps: instantiate and reduce the argument of the application, push out the resulting set braces by distributing them in the terms and only when none of the previous reductions is possible, use the evaluation rule *Fire*. We can easily express this strategy by imposing a simple condition for the application of the evaluation rule *Fire*.

Definition 4.6 We call *ConfStratStrict* the strategy which consists in applying the evaluation rule *Fire* to a redex $[l \to r](t)$ only if the term t is a first order ground term.

Proposition 4.7 When using the evaluation strategy *ConfStratStrict*, the ρ-calculus is confluent.

Proof. We consider the parallelization of the relation induced by the evaluation rules *Fire* and *Congruence* on one hand and the relation induced by the other rules of the calculus on the other hand. We show the confluence of the two relations and then use Yokouchi's Lemma [60] to prove the strong confluence of the relation obtained by combining the former relations. This latter relation is the transitive closure of the relation induced by the evaluation rules *Fire* and *Congruence*, and the evaluation rules handling sets.

The Yokouchi Lemma can be easily proved due to the strict conditions on the application of the rule *Fire* and thus to the absence of interaction between the evaluation rules of the calculus. □

The strategy *ConfStratStrict* is quite restrictive and we would like to define a general strategy that becomes trivial (*i.e.* imposes no restriction) when restricted to some simpler calculi, as the λ-calculus.

A confluent strategy emerges from the above counterexamples and allows the application of the evaluation rule *Fire* only if a possible failure in the matching is preserved by the subsequent ρ-reductions and if the argument of the application cannot be reduced to an empty set or to a set having more than one element. Such a generic strategy consists in applying the evaluation rule *Fire* to a redex $[l \to r](t)$ only if:

- $t \in \mathcal{T}(\mathcal{F})$ is a first order ground term

or

- the term t is such that if the matching $l \ll^?_\emptyset t$ fails then, for all term t' obtained by instantiating or reducing t, the matching $l \ll^?_\emptyset t'$ fails, and

- the term t cannot be reduced to an empty set or to a set having more than one element.

If we consider an instance of the ρ-calculus such that all the sets are singletons and all the applications are of the form $[x \to u](v)$ then, all the above conditions are always satisfied. Hence, we can say that in this case the previous strategy is equivalent to the strategy $\mathcal{ALL}$, *i.e.* it imposes no restriction on the reductions. One can notice that the ρ_λ-terms satisfy the previous conditions and thus, such a strategy imposes no restrictions on the reductions of this instance of the ρ-calculus.

The conditions imposed for the generic strategy when the term t is not a first order ground term are clearly not appropriate for an implementation of the ρ-calculus and thus, we must define operational strategies guaranteeing the confluence of the calculus. These strategies will impose some decidable conditions that correspond to (and imply) the ones proposed above.

We introduce in what follows a more operational and more restrictive strategy definition guaranteeing the matching *"coherence"* by imposing structural conditions on the terms l and t involved in a matching problem $l \ll^?_\emptyset t$. In order to ensure the matching failure preservation by the ρ-reductions, the failure must be generated only by different first order symbols in the corresponding positions of the two terms l and t. This property is always verified if the two terms are first order terms but an additional condition must be imposed if the term t contains ρ-calculus specific operators, as the abstraction or the application.

Definition 4.8 A ρ-term l *weakly subsumes* a ρ-term t if

$$\forall p \in \mathcal{FP}os(l) \cap \mathcal{P}os(t) \Rightarrow t(p) \in \mathcal{F}$$

Thus, a ρ-term l *weakly subsumes* a ρ-term t if for any functional position of the term l, either this position is not a position of the term t, or it is a functional position of the term t.

Remark 4.9 If $l \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ weakly subsumes t then, for any non-functional position (*i.e.* the position of a variable, an application, an abstraction or a set) in t, the corresponding position in l, if it exists, is a variable position. Thus, if the top position of t is not a functional position, then l is a variable.
One can notice that if a first order term l subsumes t, then l weakly subsumes t.

Example 4.10 The term $h(a, y, c)$ weakly subsumes the term $g(b, [x \to x](c))$ and the term $f(a)$ weakly subsumes the term $g(b, [x \to x](c))$. The term $g(a, y)$ weakly subsumes the term $g(b, [x \to x](c))$ while the term $f(a)$ does not weakly subsumes $f([x \to x](c))$.

Definition 4.11 We call *ConfStrat* the strategy which consists in applying the evaluation rule *Fire* to a redex $[l \to r](t)$ only if:

- $t \in \mathcal{T}(\mathcal{F})$ is a first order ground term

or

- the term $l \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ is linear and l weakly subsumes t, and
- the term t contains no set with more than one element and no empty set, and

- for all sub-term $[u \to w](v)$ of t, u subsumes v, and
- the term t contains no sub-term of the form $[u](v)$ where u is not an abstraction.

One should notice that the conditions imposed by the strategy *ConfStrat* are decidable even if the term t is not a first order ground term. One can clearly decide if a term is of the form $[u](v)$ or $[u \to w](v)$ as well as the number of elements of a finite set. The condition that l weakly subsumes t is simply a condition on the symbols on the same positions of the two terms and since matching is syntactic, then the subsumption condition is also decidable. Consequently, all the conditions used in the strategy *ConfStrat* are decidable.

The condition forbidding sub-terms of t of the form $[u](v)$ if u is not a rewrite rule is imposed in order to prevent the application of the evaluation rule $CongFail$ leading to an empty set result. If one considers a version of the ρ-calculus without the evaluation rules $Congruence$ then, this last condition is no longer necessary in the strategy *ConfStrat*. Hence, all the terms of the representation of the λ-calculus in the ρ-calculus trivially satisfy the above conditions and in this case the strategy *ConfStrat* is equivalent to the strategy $\mathcal{ALL}$.

Proposition 4.12 When using the evaluation strategy *ConfStrat*, the ρ-calculus is confluent.

PROOF. Starting from the evaluation rule $Fire$ expressed as a conditional rule guarded by the conditions defined in the strategy *ConfStrat* we define the relation $FireCong$ induced by this latter rule and the $Congruence$ rules. The other evaluation rules of the calculus induce a second relation called Set.

We denote by $\longrightarrow_F$ and $\longrightarrow_S$ respectively, the compatible (context) closures of these two relations, and by $\stackrel{*}{\longrightarrow}_S$ the reflexive and transitive closure of $\longrightarrow_S$.

We prove the confluence of the relation $\stackrel{*}{\longrightarrow}_S\longrightarrow_F\stackrel{*}{\longrightarrow}_S$ and we use an approach similar to the one followed in [6] for proving the confluence of $\lambda_{\Uparrow}$.

Thus, we have to prove the strong confluence of the relation $\longrightarrow_F$, the confluence and termination of $\longrightarrow_S$ and the compatibility between the two relations (*i.e.* Yokouchi's Lemma.

Using a polynomial interpretation we show that $\longrightarrow_S$ terminates and by analyzing the induced critical pairs we obtain the local confluence and consequently, the confluence of this relation.

The relation $\longrightarrow_F$ is not strongly confluent but we define the parallel version of this relation in the style of *Tait & Martin-Löf*. We denote this relation by $\longrightarrow_{F_\|}$ and we show that is strongly confluent.

The Yokouchi Lemma is proved using the conditions imposed on the application of the rule $Fire$. We obtain thus the strong confluence of the relation $\stackrel{*}{\longrightarrow}_S\longrightarrow_{F_\|}\stackrel{*}{\longrightarrow}_S$ and since this latter relation is the transitive closure of the relation $\stackrel{*}{\longrightarrow}_S\longrightarrow_F\stackrel{*}{\longrightarrow}_S$ we deduce the confluence of the calculus.

The proof is presented in full detail in [8]. □

The relatively restrictive conditions imposed in strategy *ConfStrat* can be relaxed at the price of the simplicity of the strategy. The conditions that we want to weaken concern on one hand, the number of elements of the sets and on the other hand, the form of the rewrite rules.

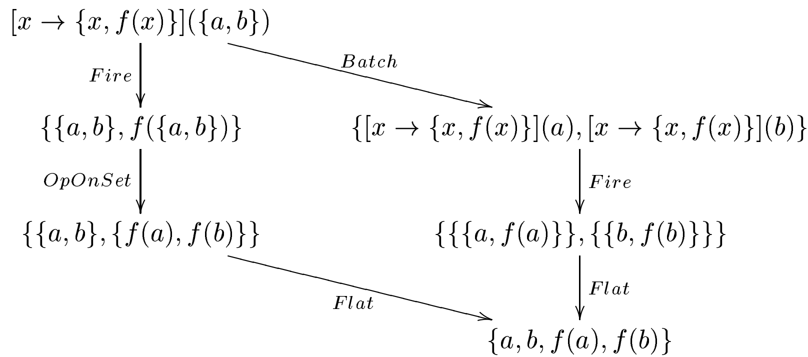
First, the absence of sets having more than one element is necessary in order to guarantee a good behavior for the non-right-linear rewrite rules. The *right-linearity*

of a rewrite rule is defined as the linearity of the right-hand side w.r.t. the variables of the left-hand side. For example, $x \to g(x,y)$ is right-linear, but $x \to g(x,x)$ is not right-linear. Moreover, the right-linearity can be imposed only to the operators different from the set symbols ($\{_\}$) and thus, the rewrite rule $x \to \{f(x), f(x)\}$ can be considered right-linear. Intuitively, we do not need to impose right-linearity for sets since, due to the evaluation rule $Flat$, they do not lead to non-convergent reductions as in Example 4.5.

Definition 4.13 The rewrite rule $l \to r$ is *hereditary right-linear* if any sub-term of r that is not a set is linear w.r.t. the free variables of l and any rewrite rule of r is hereditary right-linear.

The application of a rewrite rule which is not hereditary right-linear to a set with more than one element can lead to non-convergent reductions, as shown in Example 4.5, but this is not the case if the applied rewrite rule is hereditary right-linear:

Example 4.14



On another hand, in order to guarantee the strict propagation of the failure, we impose that the evaluation rule $Fire$ is applied only if the argument of the application is not an empty set and it cannot lead to an empty set. In Example 4.4 we can notice that the free variables of the left-hand side of the rewrite rule are not preserved in the right-hand side of the rule. If the rewrite rule $l \to r$ of the application preserves the variables of the left-hand side in the right-hand side (*e.g.* $x \to x$), the application of a substitution replacing one of these variables with an empty set (*e.g.* $\{x/\emptyset\}$) to r leads to a term containing $\emptyset$ and thus, which is possibly reduced to $\emptyset$.

We define thereafter more formally the rewrite rules preserving the variables and we present a new strategy defined using this property. First, we introduce a concept similar to that of free variable but, by considering this time the not-deterministic nature of the sets.

Definition 4.15 The set of *present variables* of a ρ-term t is denoted by $PV(t)$ and is defined by:

1. if $t = x$ then $PV(t) = \{x\}$,
2. if $t = \{u_1, \ldots, u_n\}$ then $PV(t) = \bigcap_{i=1\ldots n} PV(u_i)$, ($PV(\emptyset) = \mathcal{X}$),
3. if $t = f(u_1, \ldots, u_n)$ then $PV(t) = \bigcup_{i=1\ldots n} PV(u_i)$, ($PV(c) = \emptyset\ if\ c \in \mathcal{T}(\mathcal{F})$),
4. if $t = [u](v)$ then $PV(t) = PV(u) \cup PV(v)$,

5. if $t = u \to v$ then $PV(t) = PV(v) \setminus FV(u)$.

The set of *free variables* of a set of ρ-terms is the union of the sets of free variables of each ρ-term while the set of *present variables* of a set of ρ-terms is the intersection of the sets of free variables of each ρ-term. We can say that a variable is *present* in a set only if it is present in all the elements of the set. For example, $PV(\{x, y, x\}) = \emptyset$ and $PV(\{x, g(x, y)\}) = \{x\}$.

Definition 4.16 We say that the ρ-rewrite rule $l \to r$ is quasi-regular if $FV(l) \subseteq PV(r)$ and any rewrite rule of r is quasi-regular.

Intuitively, to each free variable of the left-hand side of a quasi-regular rewrite rule corresponds, in a deterministic way, a free variable in the right-hand side of the rule. For any set ρ-term in the right-hand side, the correspondence with the free variables of the left-hand side should be verified for each element of the set.

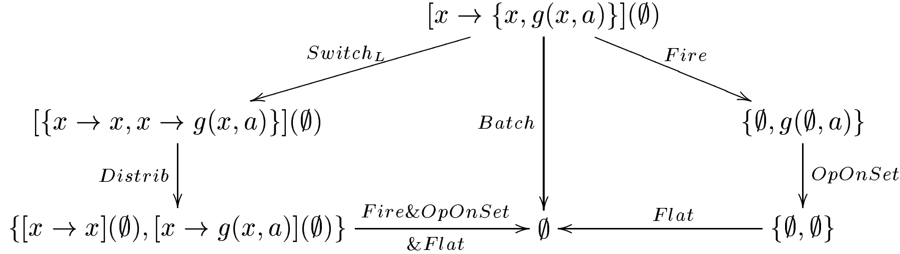
Example 4.17 The rewrite rule $x \to g(x, y)$ is quasi-regular while the rewrite rule $x \to \{x, y\}$ is non-quasi-regular.

The rewrite rule $\{f(x), g(x, x)\} \to x$ is quasi-regular while $\{f(x), g(x, y)\} \to x$ is non-quasi-regular. If the definition of quasi-regular rewrite rules had asked for the condition $PV(l) \subseteq PV(t)$ instead, then the second rewrite rule would have become quasi-regular as well. This is not desirable since the rewrite rule $\{f(x), g(x, y)\} \to x$ reduces to $\{f(x) \to x, g(x, y) \to x\}$ and only the first one is quasi-regular.

In the particular case of the ρ-calculus, since the left-hand side of a rewrite rule $l \to r$ must be a first-order term (*i.e.* $l \in \mathcal{T}(\mathcal{F}, \mathcal{X})$), we have $FV(l) = PV(l) = \mathcal{V}ar(l)$ and thus the condition from Definition 4.16 can be changed to $\mathcal{V}ar(l) \subseteq PV(t)$.

Let us consider the application a quasi-regular rewrite rule $l \to r$ to a term t giving as result the term $\{\sigma r\}$, where σ is the matching substitution between l and t. If $\emptyset$ is a sub-term of t and if l weakly subsumes r, then $\emptyset$ is in σ. Since the rewrite rule is quasi-regular, we have $\mathcal{D}om(\sigma) \subseteq PV(r)$ and thus, we are sure that $\emptyset$ is a sub-term of σr. Furthermore, if $\emptyset$ instantiated a variable of a set in σr then it is present in all the elements of the set and thus, we avoid non-confluent results as the ones in Example 4.4.

Example 4.18 A quasi-regular rule applied to $\emptyset$ gives only one result:



while a non-quasi-regular one yields two different results as shown in Example 4.4.

One should notice that if a rewrite rule $l \to r$ is reduced by the evaluation rule *Switch_R* to a set of rewrite rules, each of these rules is quasi-regular and thus the strict propagation of the empty set is ensured on all the right-hand sides of the obtained rewrite rules.

Definition 4.19 We call *ConfStratLin* the strategy which consists in applying the evaluation rule *Fire* to a redex $[l \to r](t)$ only if $t \in \mathcal{T}(\mathcal{F})$ is a first order ground term *or*:

- the term $l \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ is linear and l weakly subsumes t,

and

- *either*
  - $l \to r$ is quasi-regular

  or
  - the term t contains no empty set, and
  - for all sub-term $[u \to w](v)$ of t, u subsumes v, and
  - the term t contains no sub-term of the form $[u](v)$ where u is not an abstraction.

and

- *either*
  - $l \to r$ is hereditary right-linear

  or
  - the term t contains no set with more than one element.

Compared to the strategy *ConfStrat* we added the possibility to test either the quasi-regular condition on the rewrite rule $l \to r$ or the conditions on the reducibility of the term t to an empty set. Moreover, if the rewrite rule is hereditary right-linear we allow arguments containing sets having more than one element. Since one can clearly decide if a rule is quasi-regular or hereditary right-linear, all the conditions used in the strategy *ConfStratLin* are decidable.

Proposition 4.20 When using the evaluation strategy *ConfStratLin*, the ρ-calculus is confluent.

PROOF. The same approach as for the strategy *ConfStrat* is used but some additional diagrams corresponding to the reductions that where not possible before are considered. These new cases are mainly introduced in the proof of Yokouchis's Lemma. The proof is detailled in [8]. □

When using a calculus integrating reduction modulo an equational theory (*e.g.* associativity and commutativity), as explained in Section 2.4, the overall confluence proof is different but uses lemmas similar to the ones of the former case. Therefore, we conjecture that Proposition 4.12 and Proposition 4.20 can be extended to a ρ_E-calculus modulo a specific decidable and finitary equational matching theory E.

5 Conclusion

We have presented the ρ_T-calculus together with some of its variants obtained as instances of the general framework. By making explicit the notion of rule, rule application and application result, the ρ_T-calculus allows us to describe in a simple yet very powerful and uniform manner algebraic and higher-order capabilities. This provides therefore a simple and natural framework for their combination.

In the ρ_T-calculus the non-determinism is handled by using sets of results and the rule application failure is represented by the empty set. Handling sets is a delicate problem and we have seen that the raw ρ-calculus, where the evaluation rules are

not guided by a strategy, is not confluent. When an appropriate but rather natural generalized call-by-value evaluation strategy is used, the calculus is confluent.

The ρ-calculus is both conceptually simple as well as quite expressive. This allows us to represent the terms and reductions from λ-calculus and rewriting. We conjecture that, following the lines of [55], it is also simple to encode other calculi of interest like the π-calculus.

Part II, is devoted to the use of an extension of the calculus powerful enough to encode rewriting strategies, conditional rewriting and to give a semantics to the ELAN language. We refer to the conclusion of *Part II* for a presentation of the ongoing and future works on the ρ-calculus.

Acknowledgments

We would like to thank Hélène Kirchner, Pierre-Etienne Moreau and Christophe Ringeissen from the Protheo Team for the useful interactions we had on the topics of this paper, Vincent van Oostrom for suggestions and pointers to the literature, Roberto Bruni and David Wolfram for their detailed and very useful comments on a preliminary version of this work and Delia Kesner for fruitful discussions. We are grateful to Luigi Liquori for many comments and exciting discussions on the ρ-calculus and its applications. Many thanks also to Thérèse Hardin and Nachum Dershowitz for their interest, encouragements and helpful suggestions for improvement. Finally special thanks are due to the referees for the very complete and careful reading of the paper as well as constructive and useful remarks.

References

[1] M. Adi and C. Kirchner. Associative commutative matching based on the syntacticity of the AC theory. In F. Baader, J. Siekmann, and W. Snyder, editors, *Proceedings 6th International Workshop on Unification, Dagstuhl (Germany)*. Dagstuhl seminar, 1992.

[2] H. P. Barendregt. *The Lambda-Calculus, its syntax and semantics.* Studies in Logic and the Foundation of Mathematics. Elsevier Science Publishers B. V. (North-Holland), Amsterdam, 1984. Second edition.

[3] P. Borovanský, C. Kirchner, H. Kirchner, and C. Ringeissen. Rewriting with strategies in ELAN: a functional semantics. *International Journal of Foundations of Computer Science*, 2001.

[4] F. Baader and T. Nipkow. *Term Rewriting and all That*. Cambridge University Press, 1998.

[5] V. Breazu-Tannen. Combining algebra and higher-order types. In *Proceedings 3rd IEEE Symposium on Logic in Computer Science, Edinburgh (UK)*, pages 82–90, 1988.

[6] P.-L. Curien, T. Hardin, and J.-J. Lévy. Confluence properties of weak and strong calculi of explicit substitutions. *Journal of the ACM*, 43(2):362–397, 1996.

[7] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[8] H. Cirstea. *Calcul de réécriture : fondements et applications.* Thèse de Doctorat d'Université, Université Henri Poincaré - Nancy I, 2000.

[9] H. Cirstea and C. Kirchner. Combining higher-order and first-order computation using ρ-calculus: Towards a semantics of ELAN. In D. Gabbay and M. de Rijke, editors, *Frontiers of Combining Systems 2*, Research Studies, ISBN 0863802524, pages 95–120. Wiley, 1999.

[10] L. Colson. Une structure de données pour le λ-calcul typé. Private Communication, 1988.

[11] D. Dill, A. Drexler, A. Hu, and C. Yang. Protocol verification as a hardware design aid. In *IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 522–525. IEEE computer society, 1992.

[12] N. Dershowitz. Computing with rewrite systems. *Information and Control*, 65(2/3):122–157, 1985.

[13] G. Dowek, T. Hardin, and C. Kirchner. Higher-order unification via explicit substitutions, extended abstract. In D. Kozen, editor, *Proceedings of LICS'95*, pages 366–374, San Diego, June 1995.

[14] G. Dowek, T. Hardin, and C. Kirchner. Theorem proving modulo. Rapport de Recherche 3400, Institut National de Recherche en Informatique et en Automatique, April 1998. `ftp://ftp.inria.fr/INRIA/publication/RR/RR-3400.ps.gz`.

[15] G. Dowek, T. Hardin, C. Kirchner, and F. Pfenning. Unification via explicit substitutions: The case of higher-order patterns. In M. Maher, editor, *Proceedings of JICSLP'96*, Bonn (Germany), September 1996. The MIT press.

[16] N. Dershowitz and J.-P. Jouannaud. Rewrite Systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 6, pages 244–320. Elsevier Science Publishers B. V. (North-Holland), 1990.

[17] G. Dowek. Third order matching is decidable. *Annals of Pure and Applied Logic*, 69:135–155, 1994.

[18] S. Eker. Associative-commutative matching via bipartite graph matching. *Computer Journal*, 38(5):381–399, 1995.

[19] S. Eker. Fast matching in combinations of regular equational theories. In J. Meseguer, editor, *Proceedings of the first international workshop on rewriting logic*, volume 4, Asilomar (California), September 1996. Electronic Notes in Theoretical Computer Science.

[20] F. Fages and G. Huet. Unification and matching in equational theories. In *Proceedings Fifth Colloquium on Automata, Algebra and Programming, L'Aquila (Italy)*, volume 159 of *Lecture Notes in Computer Science*, pages 205–220. Springer-Verlag, 1983.

[21] J. Gallier and V. Breazu-Tannen. Polymorphic rewriting conserves algebraic strong normalization and confluence. In *16th Colloquium Automata, Languages and Programming*, volume 372 of *Lecture Notes in Computer Science*, pages 137–150. Springer-Verlag, 1989.

[22] J. A. Goguen, C. Kirchner, H. Kirchner, A. Mégrelis, J. Meseguer, and T. Winkler. An introduction to OBJ-3. In J.-P. Jouannaud and S. Kaplan, editors, *Proceedings 1st International Workshop on Conditional Term Rewriting Systems, Orsay (France)*, volume 308 of *Lecture Notes in Computer Science*, pages 258–263. Springer-Verlag, July 1987. Also as internal report CRIN: 88-R-001.

[23] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1989.

[24] G. Huet and B. Lang. Proving and applying program transformations expressed with second-order patterns. *Acta Informatica*, 11:31–55, 1978.

[25] J. R. Hindley and J. P. Seldin. *Introduction to Combinators and Lambda-calculus*. Cambridge University, 1986.

[26] G. Huet. A mechanization of type theory. In *Proceeding of the third international joint conference on artificial intelligence*, pages 139–146, 1973.

[27] G. Huet. *Résolution d'equations dans les langages d'ordre 1,2, ...,ω*. Thèse de Doctorat d'Etat, Université de Paris 7 (France), 1976.

[28] J.-P. Jouannaud and H. Kirchner. Completion of a set of rules modulo a set of equations. *SIAM Journal of Computing*, 15(4):1155–1194, 1986. Preliminary version in Proceedings 11th ACM Symposium on Principles of Programming Languages, Salt Lake City (USA), 1984.

[29] J.-P. Jouannaud and C. Kirchner. Solving equations in abstract algebras: a rule-based survey of unification. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic. Essays in honor of Alan Robinson*, chapter 8, pages 257–321. The MIT press, Cambridge (MA, USA), 1991.

[30] J.-P. Jouannaud and M. Okada. Abstract data type systems. *Theoretical Computer Science*, 173(2):349–391, 28 February 1997.

[31] G. Kahn. Natural semantics. Technical Report 601, INRIA Sophia-Antipolis, February 1987.

[32] D. Kesner. *La définition de fonctions par cas à l'aide de motifs dans des langages applicatifs*. PhD thesis, Université de Paris XI, December 1993.

[33] C. Kirchner and H. Kirchner. Rewriting, solving, proving. A preliminary version of a book available at `www.loria.fr/~ckirchne/rsp.ps.gz`, 1999.

[34] C. Kirchner, H. Kirchner, and M. Rusinowitch. Deduction with symbolic constraints. *Revue d'Intelligence Artificielle*, 4(3):9–52, 1990. Special issue on Automatic Deduction.

[35] C. Kirchner, H. Kirchner, and M. Vittek. Designing constraint logic programming languages using computational systems. In P. Van Hentenryck and V. Saraswat, editors, *Principles and Practice of Constraint Programming. The Newport Papers.*, chapter 8, pages 131–158. The MIT press, 1995.

[36] J. W. Klop. Term Rewriting Systems. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 1, chapter 6. Oxford University Press, 1990.

[37] C. Kirchner and C. Ringeissen. Rule-Based Constraint Programming. *Fundamenta Informaticae*, 34(3):225–262, September 1998.

[38] J. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory reduction systems: introduction and survey. *Theoretical Computer Science*, 121:279–308, 1993.

[39] J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.

[40] R. Milner. A proposal for standard ML. In *Proceedings ACM Conference on LISP and Functional Programming*, 1984.

[41] D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. In P. Schroeder-Heister, editor, *Extensions of Logic Programming: International Workshop, Tübingen, Germany, December 1989*, volume 475 of *Lecture Notes in Computer Science*, pages 253–281. Springer-Verlag, 1991.

[42] MuPAD Group, Benno Fuchssteiner et al. *MuPAD User's Manual - MuPAD Version 1.2.2.* John Wiley and sons, Chichester, New York, first edition, march 1996. includes a CD for Apple Macintosh and UNIX.

[43] T. Nipkow. Combining matching algorithms: The regular case. In N. Dershowitz, editor, *Proceedings 3rd Conference on Rewriting Techniques and Applications, Chapel Hill (N.C., USA)*, volume 355 of *Lecture Notes in Computer Science*, pages 343–358. Springer-Verlag, April 1989.

[44] T. Nipkow and C. Prehofer. Higher-order rewriting and equational reasoning. In W. Bibel and P. Schmitt, editors, *Automated Deduction — A Basis for Applications. Volume I: Foundations.* Kluwer, 1998.

[45] M. J. O'Donnell. *Computing in Systems Described by Equations*, volume 58 of *Lecture Notes in Computer Science*. Springer-Verlag, 1977.

[46] M. Okada. Strong normalizability for the combined system of the typed λ calculus and an arbitrary convergent term rewrite system. In G. H. Gonnet, editor, *Proceedings of the ACM-SIGSAM 1989 International Symposium on Symbolic and Algebraic Computation: ISSAC '89 / July 17–19, 1989, Portland, Oregon*, pages 357–363, New York, NY 10036, USA, 1989. ACM Press.

[47] V. Padovani. *Filtrage d'ordre supérieur.* Thèse de Doctorat d'Université, Université Paris VII, 1996.

[48] V. Padovani. Decidability of fourth-order matching. *Mathematical Structures in Computer Science*, 3(10):361–372, June 2000.

[49] B. Pagano. X.R.S : Explicit Reduction Systems - A First-Order Calculus for Higher-Order Calculi. In C. Kirchner and H. Kirchner, editors, *15th International Conference on Automated Deduction*, LNAI 1421, pages 72–87, Lindau, Germany, July 5–July 10, 1998. Springer-Verlag.

[50] S. Peyton-Jones. *The implementation of functional programming languages.* Prentice Hall, Inc., 1987.

[51] Protheo Team. The ELAN home page. WWW Page, 2001. `http://elan.loria.fr`.

[52] C. Ringeissen. Combining Decision Algorithms for Matching in the Union of Disjoint Equational Theories. *Information and Computation*, 126(2):144–160, May 1996.

[53] A. van Deursen. An Overview of ASF+SDF. In *Language Prototyping*, pages 1–31. World Scientific, 1996. ISBN 981-02-2732-9.

[54] M. van den Brand, A. van Deursen, P. Klint, S. Klusener, and E. A. van der Meulen. Industrial applications of ASF+SDF. In M. Wirsing and M. Nivat, editors, *AMAST '96*, volume 1101 of *Lecture Notes in Computer Science*, pages 9–18. Springer-Verlag, 1996.

[55] P. Viry. Input/Output for ELAN. In J. Meseguer, editor, *Proceedings of the first international workshop on rewriting logic*, volume 4 of *Electronic Notes in TCS*, Asilomar (California), September 1996.

[56] M. Vittek. ELAN*: Un cadre logique pour le prototypage de langages de programmation avec contraintes.* Thèse de Doctorat d'Université, Université Henri Poincaré – Nancy 1, October 1994.

[57] V. van Oostrom. Lambda calculus with patterns. Technical report, Vrije Universiteit, Amsterdam, November 1990.

[58] D. A. Wolfram. *The Clausal Theory of Types*, volume 21 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1993.

[59] S. Wolfram. *The Mathematica Book*, chapter Patterns, Transformation Rules and Definitions. Cambridge University Press, 1999. ISBN 0-521-64314-7.

[60] H. Yokouchi and T. Hikita. A rewriting system for categorical combinators with multiple arguments. *SIAM Journal of Computing*, 19(1), February 1990.

Received October 1, 2000. Revised: January 26, 2001, February 9, 2001

The rewriting calculus — Part II

HORATIU CIRSTEA, *LORIA and INRIA, Campus Scientifique, BP 239, 54506 Vandoeuvre-lès-Nancy, France, E-mail: Horatiu.Cirstea@loria.fr.*

CLAUDE KIRCHNER, *LORIA and INRIA, Campus Scientifique, BP 239, 54506 Vandoeuvre-lès-Nancy, France, E-mail: Claude.Kirchner@loria.fr.*

Abstract

The ρ-calculus integrates in a uniform and simple setting first-order rewriting, λ-calculus and non-deterministic computations. Its abstraction mechanism is based on the rewrite rule formation and its main evaluation rule is based on matching modulo a theory T.

We have seen in the first part of this work the motivations, definitions and basic properties of the ρ-calculus. This second part is first devoted to the use of an extension of the ρ-calculus for encoding a (conditional) rewrite relation. This extension is based on the *first* operator whose purpose is to detect rule application failure. It allows us to express recursively rule application and therefore to encode strategy based rewriting processes. We then use this extended calculus to give an operational semantics to ELAN programs.

We conclude with an overview of ongoing and future works on ρ-calculus.

Keywords: rewriting, strategy, non-determinism, matching, rewriting-calculus, lambda-calculus, rule based language.

1 Introduction

This is the second part of the rewriting calculus description, study and applications. In all the paper, we refer to the first part of this work as *Part I*.

As we have seen in *Part I*, we can encode in ρ-calculus the representation of a finite derivation. But we need more since we want to be able the represent also in the calculus the generic search for normalization derivations, when they exist. More generally, we want to have a formal representation of rewriting strategies like the ones used in ELAN [25].

To this end we extend the calculus with a *first* operator whose purpose is to detect rule application failure. This extension allows us to express recursively rule application and therefore to encode strategy based rewriting processes.

We then extend the ρ-encoding of conditional rewriting to more complicated rules like the conditional rewrite rules with local assignments from the ELAN language. The non-determinism that in ELAN is handled mainly by two basic strategy operators is represented in the ρ-calculus by means of sets. We show finally how the ρ-calculus provides a semantics to ELAN programs.

This paper is structured as follows. In Section 2 we extend the basic ρ-calculus with a new operator and define term traversal and fixed-point operators using the existing ρ-operators.

 ©Oxford University Press

The encoding of non-conditional and conditional term rewriting by using the ρ-operators defined in Section 2 is presented in Section 3. The calculus is finally used in Section 4 in order to give an operational semantics to the rules used in the ELAN language.

We conclude by providing some of the research directions that are of main interest in the development of this formalism and in the context of ELAN, and more generally of rewrite based languages as ASF+SDF [20], ML [23], Maude [7], Stratego [28] or CafeOBJ [17].

2 Recursion and term traversal operators

In *Part I* we have shown that for any reduction in a rewrite theory there exists a corresponding reduction in the ρ-calculus: if the term u reduces to the term v in a rewrite theory $\mathcal{R}$ we can build a ρ-term $\xi_{\mathcal{R}}(u)$ that reduces to the term $\{v\}$. The method used for constructing the term $\xi_{\mathcal{R}}(u)$ depends on all the reduction steps from u to v in the theory $\mathcal{R}$: $\xi_{\mathcal{R}}(u)$ is a representation in the ρ-calculus of the derivation trace. We want to go further on and to give a method for constructing a term $\xi_{\mathcal{R}}(u)$ without knowing a priori the derivation from u to v. Hence we want to answer to the following question:

Given a rewrite theory $\mathcal{R}$ does there exist a ρ-term $\xi_{\mathcal{R}}$ such that for any term u, if u reduces to the term v in the rewrite theory $\mathcal{R}$ then $[\xi_{\mathcal{R}}](u)$ ρ-reduces to a set containing the term v?

This means that we wish to describe in the ρ-calculus reduction strategies and, mainly, normalization strategies. This will allow us to get, in particular, a natural encoding of normal conditional term rewriting. Therefore, we want to answer the more specific question:

Given a rewrite theory $\mathcal{R}$ does there exist a ρ-term $\xi_{\mathcal{R}}$ such that for any term u if u normalizes to the term v in the rewrite theory $\mathcal{R}$ then $[\xi_{\mathcal{R}}](u)$ ρ-reduces to a set containing the term v?

The definition of normalization strategies is in general done at the *meta-level* while the ρ-calculus allows us to represent such derivations at the *object level*. We have shown in *Part I* that the $\rho_{\emptyset}$-calculus contains the λ-calculus and thus, any computable function as the normalization one is expressible in the formalism. What we bring here, because of the matching power and of the use of non-determinism, is an increased ease in the expression of such functions together with their expression in a uniform formalism combining standard rewrite techniques and higher-order behaviors.

When computing the normal form of a term u w.r.t. a rewrite system $\mathcal{R}$, the rewrite rules are applied *repeatedly* at *any position* of a term u until no rule from $\mathcal{R}$ is *applicable*. Hence, the ingredients needed for defining such a strategy are:

- an iteration operator that applies *repeatedly* a set of rewrite rules,
- a term traversal operator that applies a rewrite rule at *any position* of a term,
- an operator testing if a set of rewrite rules is *applicable* to a term.

In what follows we describe how the operators with the above functionalities can be defined in the ρ-calculus. We start with some auxiliary operators and afterwards, we introduce the ρ-operators that correspond to the functionalities listed above.

2.1 Some auxiliary operators

First, we define three auxiliary operators that will be used in the next sections. These operators are just aliases used to define more complex ρ-terms and are used for giving more compact and clear definitions for the recursion operators.

The first of these operators is the *identity* (denoted id) that applied to any ρ-term t evaluates to the singleton containing this term, that is $[id](t) \longrightarrow_\rho \{t\}$. The ρ-term id is nothing else but the rewrite rule $x \to x$:

$$id \triangleq x \to x.$$

In a similar way we can define the strategy $fail$ which always fails, (*i.e.* applied to any term, leads to $\emptyset$):

$$fail \triangleq x \to \emptyset.$$

The third one is the binary operator ";" that represents the sequential application of two ρ-terms. A ρ-term of the form $[u;v](t)$ represents the application of the term v to the result of the application of u to t. Therefore, we define the operator ";" by:

$$u;v \triangleq x \to [v]([u](x)).$$

In the following sections we generally employ the abbreviations of these operators and not their expanded form but we sometimes show the corresponding reductions.

2.2 The $first$ operator

We introduce now a new operator, similar to the THEN operator for combining tactics and already present in LCF [18]. Its role is to select between its arguments the first one that applied to a given ρ-term does not evaluate to $\emptyset$. If all the arguments evaluate to $\emptyset$ then the final result of the evaluation is $\emptyset$. The evaluation rules describing the $first$ operator and the auxiliary operator $\langle _, \dots, _ \rangle$ are presented in Figure 1. We do not know currently how to express these operators in the basic ρ-calculus and we conjecture that this is not possible.

For simplicity, we considered that the operators $first$ and $\langle\rangle$ are of variable arity but similar binary operators can be used instead.

The application of a ρ-term $first(s_1, \dots, s_n)$ to a term t returns the result of the first "successful" application of one of its arguments to the term t. Hence, if $[s_i](t)$ evaluates to $\emptyset$ for $i = 1, \dots, k-1$, and $[s_k](t)$ does not evaluate to $\emptyset$, then $[first(s_1, \dots, s_n)](t)$ evaluates to the same term as the term $[s_k](t)$.

If the evaluation of the terms $[s_i](t)$, $i = 1, \dots, k-1$, leads to $\emptyset$ and the evaluation of $[s_k](t)$ does not terminate then the evaluation of the term $[first(s_1, \dots, s_n)](t)$ does not terminate.

Definition 2.1 The set of ρ^{1st}-terms extends the set $\varrho(\mathcal{F}, \mathcal{X})$ of basic ρ-terms, with the following two rules:

$First$	$[first(s_1,\dots,s_n)](t)$	$\Longrightarrow$	$\langle [s_1](t),\dots,[s_n](t)\rangle$
$FirstFail$	$\langle \emptyset, t_1,\dots,t_n\rangle$	$\Longrightarrow$	$\langle t_1,\dots,t_n\rangle$
$FirstSuccess$	$\langle t, t_1,\dots,t_n\rangle$	$\Longrightarrow$	$\{t\}$ if t contains no redexes, no free variables and is not $\emptyset$
$FirstSingle$	$\langle\rangle$	$\Longrightarrow$	$\emptyset$

FIG. 1. The $first$ operator

- if $t_1,\dots,t_n$ are ρ-terms then $first(t_1,\dots,t_n)$ is a ρ-term,
- if $t_1,\dots,t_n$ are ρ-terms then $\langle t_1,\dots,t_n\rangle$ is a ρ-term.

This set of terms is denoted by $\varrho^{1st}(\mathcal{F},\mathcal{X})$.

We define now the ρ_T^{1st}-calculus by considering the new operators and the corresponding evaluation rules presented in Figure 1:

Definition 2.2 Given a set $\mathcal{F}$ of function symbols, a set $\mathcal{X}$ of variables, a theory T on $\varrho^{1st}(\mathcal{F},\mathcal{X})$ terms having a decidable matching problem, we call ρ_T^{1st}-calculus a calculus defined by:

- a non-empty subset $\varrho_{_}^{1st}(\mathcal{F},\mathcal{X})$ of the $\varrho^{1st}(\mathcal{F},\mathcal{X})$ terms,
- the (higher-order) substitution application to terms as defined in *Part I*,
- a theory T,
- the set of evaluation rules $\mathcal{E}_{\rho^{1st}}$: *Fire*, *Cong*, *CongFail*, *Distrib*, *Batch*, $Switch_L$, $Switch_R$, *OpOnSet*, *Flat*, *First*, *FirstFail*, *FirstSuccess*, *FirstSingle*,
- an evaluation strategy $\mathcal{S}$ that guides the application of the evaluation rules.

In what follows we consider the ρ^{1st}-calculus, *i.e.* the ρ_T^{1st}-calculus with a syntactic matching and whose rewrite rules are restricted to be of the form $u \to v$ where u is a first-order term.

The following examples present the evaluation of some ρ^{1st}-terms containing the operators of the extended calculus.

Example 2.3 The non-deterministic application of one of the rules $a \to b$, $a \to c$, $a \to d$ to the term a is represented in the ρ-calculus by the application $[\{a \to b, a \to c, a \to d\}](a)$. This last ρ-term is reduced to the term $\{b,c,d\}$ which represents a non-deterministic choice among the three terms. If we want to apply the above rules in a deterministic way and in the specified order, we use the ρ-term $[first(a \to b, a \to c, a \to d)](a)$ with, for example, the reduction:

$$
\begin{array}{ll}
 & [first(a \to b, a \to c, a \to d)](a) \\
\longrightarrow_{First} & \langle [a \to b](a), [a \to c](a), [a \to d](a)\rangle \\
\longrightarrow_{Fire} & \langle \{b\}, [a \to c](a), [a \to d](a)\rangle \\
\longrightarrow_{FirstSuccess} & \{\{b\}\} \\
\longrightarrow_{Flat} & \{b\}
\end{array}
$$

We can notice that even if all the rewrite rules can be applied successfully (*i.e.* no empty set) to the term a, the final result is given by the first tried rewrite rule.

Example 2.4 We consider now the case where some of the rules given in argument to $first$ lead to an empty set result:

$$\begin{array}{ll}
 & [first(a \to b, b \to c, a \to d)](b) \\
\longrightarrow_{First} & \langle [a \to b](b), [b \to c](b), [a \to d](b) \rangle \\
\longrightarrow_{Fire} & \langle \emptyset, [b \to c](b), [a \to d](b) \rangle \\
\longrightarrow_{FirstFail} & \langle [b \to c](b), [a \to d](b) \rangle \\
\longrightarrow_{Fire} & \langle \{c\}, [a \to d](b) \rangle \\
\longrightarrow_{FirstSuccess} & \{\{c\}\} \\
\longrightarrow_{Flat} & \{c\}
\end{array}$$

Example 2.5 If none of the rules given in argument to $first$ is applied successfully, the result is obviously the empty set:

$$\begin{array}{ll}
 & [first(a \to b, a \to c, a \to d)](b) \\
\longrightarrow_{First} & \langle [a \to b](b), [a \to c](b), [a \to d](b) \rangle \\
\stackrel{*}{\longrightarrow}_{Fire} & \langle \emptyset, \emptyset, \emptyset \rangle \\
\stackrel{*}{\longrightarrow}_{FirstFail} & \langle \rangle \\
\longrightarrow_{FirstSingle} & \emptyset
\end{array}$$

The operator $first$ does not test explicitly the applicability of a term (rule) to another term but allows us to recover from a failure and continue the evaluation. For example, we can define a term

$$try(s) \triangleq first(s, id)$$

that applied to the term t evaluates to the result of $[s](t)$, if $[s](t)$ does not evaluate to $\emptyset$ and to $\{t\}$, if $[s](t)$ evaluates to $\emptyset$.

2.3 Term traversal operators

Let us now define operators that apply a ρ-term at some position of another ρ-term. The first step is the definition of two operators that push the application of a ρ-term one level deeper on another ρ-term. This is already possible in the ρ-calculus due to the rule $Cong$ but we want to define a generic operator that applies a ρ-term r to the sub-terms u_i, $i = 1 \dots n$, of a term of the form $F(u_1, \dots, u_n)$ independently on the head symbol F.

To this end, we define two term traversal operators, $\Phi(r)$ and $\Psi(r)$, whose behavior is described by the rules in Figure 2. These operators are inspired by the operators of the *System S* described in [27].

The application of the ρ-term $\Phi(r)$ to a term $t = f(u_1, \dots, u_n)$ results in the successful application of the term r to one of the terms u_i. More precisely, r is applied to the first u_i, $i = 1, \dots, n$ such that $[r](u_i)$ does not evaluate to the empty set. If there *exists no* such u_i and in particular, if t is a function with no arguments (t is a constant), then the term $[\Phi(r)](t)$ reduces to the empty set:

$$[\Phi(r)](c) \longrightarrow_{TraverseSeq} \langle \{\} \rangle \longrightarrow_{FirstFail} \langle \rangle \longrightarrow_{FirstSingle} \emptyset$$

$$\begin{array}{lll} \mathit{TraverseSeq} & [\Phi(r)](f(u_1,\ldots,u_n)) & \Longrightarrow \\ & \langle\{f([r](u_1),\ldots,u_n)\},\ldots,\{f(u_1,\ldots,[r](u_n))\}\rangle & \\ \mathit{TraversePar} & [\Psi(r)](f(u_1,\ldots,u_n)) & \Longrightarrow \quad \{f([r](u_1),\ldots,[r](u_n))\} \end{array}$$

FIG. 2. The term traversal operators of the ρ_T-calculus

When the ρ-term $\Psi(r)$ is applied to a term $t = f(u_1,\ldots,u_n)$ the term r is applied to all the arguments u_i, $i = 1,\ldots,n$ if *for all* i, $[r](u_i)$ does not evaluate to $\emptyset$. If there exists an u_i such that $[r](u_i)$ reduces to $\emptyset$, then the result is the empty set. If we apply $\Psi(r)$ to a constant c, since there are no sub-terms the term $[\Psi(r)](c)$ reduces to $\{c\}$:

$$[\Psi(r)](c) \longrightarrow_{TraversePar} \{c\}$$

If we consider a ρ-calculus with a finite signature $\mathcal{F}$ and if we denote by $\mathcal{F}_0 = \{c_1,\ldots,c_n\}$ the set of constant function symbols and by $\mathcal{F}_+ = \{f_1,\ldots,f_m\}$ the set of function symbols with arity at least one, the two term traversal operators can be expressed in the ρ-calculus by some appropriate ρ-terms.

If the following two definitions are considered

$$\begin{aligned} \Phi'(r) \triangleq\ & first(f_1(r,id,\ldots,id),\ldots,f_1(id,\ldots,id,r),\ldots, \\ & f_m(r,id,\ldots,id),\ldots,f_m(id,\ldots,id,r)) \end{aligned}$$

$$\Psi(r) \triangleq \{c_1,\ldots,c_n,f_1(r,\ldots,r),\ldots,f_m(r,\ldots,r)\}$$

with $c_i \in \mathcal{F}_0$, $i = 1,\ldots,n$, and $f_j \in \mathcal{F}_+$, $j = 1,\ldots,m$, we obtain the following two reductions,

$$\begin{array}{ll} & [\Phi'(r)](f_k(u_1,\ldots,u_p)) \\ \triangleq & [first(f_1(r,id,\ldots,id),\ldots,f_m(id,\ldots,id,r))](f_k(u_1,\ldots,u_p)) \\ \longrightarrow_{First} & \langle[f_1(r,id,\ldots,id)](f_k(u_1,\ldots,u_p)),\ldots,[f_m(id,\ldots,id,r)](f_k(u_1,\ldots,u_p))\rangle \\ \stackrel{*}{\longrightarrow}_{Cong} & \langle\emptyset,\ldots,\emptyset,\{f_k([r](u_1),\ldots,u_p)\},\ldots,\{f_k(u_1,\ldots,[r](u_p))\},\emptyset,\ldots,\emptyset\rangle \\ \stackrel{*}{\longrightarrow}_{FirstFail} & \langle\{f_k([r](u_1),\ldots,u_p)\},\ldots,\{f_k(u_1,\ldots,[r](u_p))\},\emptyset,\ldots,\emptyset\rangle \end{array}$$

and

$$\begin{array}{ll} & [\Psi(r)](f_k(u_1,\ldots,u_p)) \\ \triangleq & [\{c_1,\ldots,c_n,f_1(r,\ldots,r),\ldots,f_m(r,\ldots,r)\}](f_k(u_1,\ldots,u_p)) \\ \longrightarrow_{Distrib} & \{[c_1](f_k(u_1,\ldots,u_p)),\ldots\ldots,[f_m(r,\ldots,r)](f_k(u_1,\ldots,u_p))\} \\ \stackrel{*}{\longrightarrow}_{Cong} & \{\emptyset,\ldots,\emptyset,\{f_k([r](u_1),\ldots,[r](u_p))\},\emptyset,\ldots,\emptyset\} \\ \stackrel{*}{\longrightarrow}_{Flat} & \{f_k([r](u_1),\ldots,[r](u_p))\} \end{array}$$

The operator Φ' does not correspond exactly to the definition from the Figure 2 but, as we have just seen above, a similar result is obtained when applying the terms $\Phi(r)$ and $\Phi'(r)$ to a term $f_k(u_1,\ldots,u_p)$.

Lemma 2.6 The term traversal operators Φ and Ψ can be expressed in the ρ^{1st}-calculus.

PROOF. If we consider $t = f_k(u_1, \ldots, u_p)$ and if for all $i = 1, \ldots, p$ we have the reductions $[r](u_i) \stackrel{*}{\longrightarrow}_\rho \emptyset$ then, according to the evaluation rules describing the behavior of $\Phi(r)$, we obtain:

$$\begin{array}{ll}
 & [\Phi(r)](f_k(u_1, \ldots, u_p)) \\
\longrightarrow_{TraverseSeq} & \langle \{f_k([r](u_1), \ldots, u_p)\}, \ldots, \{f_k(u_1, \ldots, [r](u_p))\}\rangle \\
\stackrel{*}{\longrightarrow}_\rho & \langle \{f_k(\emptyset, \ldots, u_p)\}, \ldots, \{f_k(u_1, \ldots, \emptyset)\}\rangle \\
\stackrel{*}{\longrightarrow}_{OpOnSet} & \langle \{\emptyset\}, \ldots, \{\emptyset\}\rangle \\
\stackrel{*}{\longrightarrow}_{Flat} & \langle \emptyset, \ldots, \emptyset\rangle \\
\stackrel{*}{\longrightarrow}_{FirstFail} & \langle\rangle \\
\longrightarrow_{FirstSingle} & \emptyset
\end{array}$$

Otherwise, if there exists an l such that $[r](u_i) \stackrel{*}{\longrightarrow}_\rho \emptyset$, $i = 1, \ldots, l-1$ and $[r](u_l) \stackrel{*}{\longrightarrow}_\rho v_l\downarrow$, with $v_l\downarrow$ a ground term containing no redex, the following reduction is obtained:

$$\begin{array}{ll}
 & [\Phi(r)](f_k(u_1, \ldots, u_p)) \\
\longrightarrow_{TraverseSeq} & \langle \{f_k([r](u_1), \ldots, u_p)\}, \ldots, \{f_k(u_1, \ldots, [r](u_p))\}\rangle \\
\stackrel{*}{\longrightarrow}_\rho & \langle \{f_k(\emptyset, \ldots, u_p)\}, \ldots, \{f_k(u_1, \ldots, v_l\downarrow, \ldots, u_p)\}, \ldots, \{f_k(u_1, \ldots, \emptyset)\}\rangle \\
\stackrel{*}{\longrightarrow}_{OpOnSet} & \langle \emptyset, \ldots, \emptyset, \{f_k(u_1, \ldots, v_l\downarrow, \ldots, u_p)\}, \emptyset, \ldots, \emptyset\rangle \\
\stackrel{*}{\longrightarrow}_{FirstFail} & \langle \{f_k(u_1, \ldots, v_l\downarrow, \ldots, u_p)\}, \emptyset, \ldots, \emptyset\rangle
\end{array}$$

Depending on the evaluation strategy, the terms following $f_k(u_1, \ldots, v_l\downarrow, \ldots, u_p)$ can be reduced or not to the empty set and we have chosen here the former alternative for a more compact representation.

Now, if we consider the definition of $\Phi'(r)$ and if for all $i = 1, \ldots, p$ we have $[r](u_i) \stackrel{*}{\longrightarrow}_\rho \emptyset$ then, we obtain:

$$\begin{array}{ll}
 & [\Phi'(r)](f_k(u_1, \ldots, u_p)) \\
\stackrel{*}{\longrightarrow}_\rho & \langle \{f_k([r](u_1), \ldots, u_p)\}, \ldots, \{f_k(u_1, \ldots, [r](u_p))\}, \emptyset, \ldots, \emptyset\rangle \\
\stackrel{*}{\longrightarrow}_\rho & \langle \{f_k(\emptyset, \ldots, u_p)\}, \ldots, \{f_k(u_1, \ldots, \emptyset)\}, \emptyset, \ldots, \emptyset\rangle \\
\stackrel{*}{\longrightarrow}_{OpOnSet} & \langle \{\emptyset\}, \ldots, \{\emptyset\}, \emptyset, \ldots, \emptyset\rangle \\
\stackrel{*}{\longrightarrow}_{Flat} & \langle \emptyset, \ldots, \emptyset, \ldots, \emptyset\rangle \\
\stackrel{*}{\longrightarrow}_{FirstFail} & \langle\rangle \\
\longrightarrow_{FirstSingle} & \emptyset
\end{array}$$

For the same term $[\Phi'(r)](f_k(u_1, \ldots, u_p))$, if it exists an l such that $[r](u_i) \stackrel{*}{\longrightarrow}_\rho \emptyset$, $i = 1, \ldots, l-1$ and $[r](u_l) \stackrel{*}{\longrightarrow}_\rho v_l\downarrow$, with $v_l\downarrow$ a ground term containing no redex, the following reduction is obtained:

$$\begin{array}{ll}
 & [\Phi'(r)](f_k(u_1, \ldots, u_p)) \\
\stackrel{*}{\longrightarrow}_\rho & \langle \{f_k([r](u_1), \ldots, u_p)\}, \ldots, \{f_k(u_1, \ldots, [r](u_p))\}, \emptyset, \ldots, \emptyset\rangle \\
\stackrel{*}{\longrightarrow}_\rho & \langle \{f_k(\emptyset, \ldots, u_p)\}, \ldots, \{f_k(u_1, \ldots, v_l\downarrow, \ldots, u_p)\}, \emptyset, \ldots, \emptyset\rangle \\
\stackrel{*}{\longrightarrow}_{OpOnSet} & \langle \{\emptyset\}, \ldots, \{\emptyset\}, \{f_k(u_1, \ldots, v_l\downarrow, \ldots, u_p)\}, \emptyset, \ldots, \emptyset\rangle \\
\stackrel{*}{\longrightarrow}_{Flat} & \langle \emptyset, \ldots, \emptyset, \{f_k(u_1, \ldots, v_l\downarrow, \ldots, u_p)\}, \emptyset, \ldots, \emptyset\rangle \\
\stackrel{*}{\longrightarrow}_{FirstFail} & \langle \{f_k(u_1, \ldots, v_l\downarrow, \ldots, u_p)\}, \emptyset, \ldots, \emptyset\rangle
\end{array}$$

We can notice that the results of the reductions for the application of a term r to the arguments of a term $f_k(u_1, \dots, u_p)$ by using the two operators, Φ and Φ', are identical. If the terms u_i, $i = 1 \dots p$, are ground terms containing no redex then, the final result of the two reductions in the case without failure is $\{f_k(u_1, \dots, v_l \downarrow, \dots, u_p)\}$.

When the operators are applied to a constant $c_k \in \mathcal{F}_0$ we obtain:

$$[\Phi'(r)](c_k) \xrightarrow{*}_\rho \langle\rangle \longrightarrow_\rho \emptyset,$$

$$[\Psi(r)](c_k) \xrightarrow{*}_\rho \{c_k\}.$$

□

2.4 Iterators

The definition of the evaluation (normalization) strategies as, for example, *top-down* or *bottom-up*, is based on the application of one term to the top position or to the deepest positions of another term.

For the moment, we have the possibility of applying a ρ-term r either to one or all the arguments u_i of a ρ-term $t = f(u_1, \dots, u_n)$, or to the sub-terms of t at an explicitly specified depth. But the depth of a term is not known *a priori* and thus, we cannot apply a term r to the deepest positions of a term t. If we want to apply the term r to the sub-terms at the maximum depth of a term t we must define a recursive operator which reiterates the application of the $\Phi(r)$ and $\Psi(r)$ terms and thus, pushes the application deeper into terms.

We start by presenting the ρ-term used for describing recursive applications in the ρ-calculus. Starting from the fixed-point combinators of the λ-calculus, we define a ρ-term which recursively applies a given ρ-term. We use the classical fixed-point combinator of the λ-calculus ([2]), $\Theta_\lambda = (A_\lambda\ A_\lambda)$ where

$$A_\lambda = \lambda xy.y(xxy)$$

and Θ_λ is called the Turing fixed-point combinator ([26]).

This term corresponds in the ρ-calculus to the ρ-term $\Theta = [A](A)$ with

$$A = x \to (y \to [y]([[x](x)](y))).$$

In λ-calculus, for any λ-term G we have the reduction

$$\Theta_\lambda\ G \xrightarrow{*}_\beta G(\Theta_\lambda\ G).$$

In ρ-calculus, we have a similar reduction

$$[\Theta](G) \xrightarrow{*}_\rho \{[G]([\Theta](G))\} \qquad (\textit{Fixed Point})$$

as this can be checked as follows:

$$\begin{array}{ll}
 & [\Theta](G) \triangleq [[A](A)](G) \triangleq [[x \to (y \to [y]([[x](x)](y)))](A)](G) \\
\longrightarrow_{Fire} & [\{y \to [y]([[A](A)](y))\}](G) \\
\longrightarrow_{Distrib} & \{[y \to [y]([[A](A)](y))](G)\} \\
\longrightarrow_{Fire} & \{\{[G]([[A](A)](G))\}\} \\
\longrightarrow_{Flat} & \{[G]([[A](A)](G))\} \\
\triangleq & \{[G]([\Theta](G))\}
\end{array}$$

We have obtained the desired result but the last application of the rule $Fire$ in the above reduction can be replaced by a reduction in the sub-term $[[A](A)](y)$. We can thus reduce $[[A](A)](y) \triangleq [[x' \to (y' \to [y']([[x'](x')](y')))](A)](y)$ to the term $\{[y]([[A](A)](y))\} \triangleq \{[y]([\Theta](y))\}$. We therefore obtain the following derivation:

$$
\begin{array}{ll}
 & [\Theta](G) \\
\xrightarrow{*}_{\rho} & \{[y \to [y]([[A](A)](y))](G)\} \triangleq \{[y \to [y]([\Theta](y))](G)\} \\
\xrightarrow{*}_{\rho} & \{[y \to [y](\{[y]([\Theta](y))\})](G)\} \\
\xrightarrow{*}_{\rho} & \{[y \to [y]([y]([\Theta](y)))](G)\} \\
\xrightarrow{*}_{\rho} & \ldots
\end{array}
$$

which does not terminate if the same redex $[\Theta](y)$ is always selected for reduction.

In an operational approach we do not want the new constructions to lead to non-terminating reductions. Since the ρ-term $[\Theta](G)$ can obviously lead to infinite reductions, a strategy should be used in order to obtain termination and thus the desired behavior.

We should thus use a strategy which applies the evaluation rules to a sub-term of the form $[\Theta](G)$ only when no other reduction is possible. From an operational point of view, this strategy is rather difficult to implement and obviously not very efficient in a calculus where the Θ term is represented by its extended form and thus, more difficult to identify. If Θ is considered as an independent ρ-term with the behavior described by an evaluation rule corresponding to the reduction ($Fixed\ Point$), the strategy suggested previously could be easily implemented.

A strategy satisfying the termination condition and easier to implement could initially apply the evaluation rules at the top positions of the terms and only when no evaluation rule can be applied at the top position, reduce the sub-terms at deeper positions. In what follows we will generally use this *outermost* strategy. It is clear that such a strategy prevents only the infinite reductions due to the operator Θ, but it cannot ensure the termination of the untyped ρ-calculus.

As we mentioned previously, the main goal of this section is the representation of normalization strategies by ρ-terms and thus, we want to describe the application of a term r to all the positions of another term t. Therefore, we must define the appropriate term G that propagates the application of a ρ-term in the sub-terms of another ρ-term.

2.4.1 Multiple applications

First, we want to define the operators $BottomUp$ and $TopDown$ describing the application of a term r to all the sub-terms of a term t starting with the deepest positions of t and respectively with the top position of t. We want thus to find a term which recursively applies the term r to all the sub-terms of t and afterwards at the top position of the result term and another term which initially applies the term r at the top position of the term t and then to the sub-terms of the result term. The term r must be applied to the sub-terms only if this application does not lead to a failure.

We propose first two "naive" definitions for the former operator and we comment the encountered problems. We analyze the obtained reductions and we define afterwards the operators describing the desired behavior.

The first natural possibility is to define the ρ-term

$$G_{sds}(r) \triangleq f \to (x \to [\Psi(f);r](x))$$

Let us consider the ρ-term SDS (for *SpreadDownSimple*),

$$SDS(r) \triangleq [\Theta](G_{sds}(r))$$

and its application to the term $t = f(t_1,\ldots,t_n)$. Then, the following derivation is obtained:

$$
\begin{array}{ll}
 & [SDS(r)](t) \triangleq [[\Theta](G_{sds}(r))](t) \\
\xrightarrow{*}_\rho & \{[[G_{sds}(r)]([\Theta](G_{sds}(r)))](t)\} \\
\triangleq & \{[[G_{sds}(r)](SDS(r))](t)\} \\
\triangleq & \{[[f \to (x \to [\Psi(f);r](x))](SDS(r))](t)\} \\
\xrightarrow{*}_\rho & \{[\{x \to [\Psi(SDS(r));r](x)\}](t)\} \\
\xrightarrow{*}_\rho & \{[\Psi(SDS(r));r](f(t_1,\ldots,t_n))\} \\
\xrightarrow{*}_\rho & \{[r]([\Psi(SDS(r))](f(t_1,\ldots,t_n)))\} \\
\xrightarrow{*}_\rho & \{[r](f([SDS(r)](t_1),\ldots,[SDS(r)](t_n)))\}
\end{array}
$$

As we can see from this derivation, the term $SDS(r)$ is recursively applied to the sub-terms of the initial term and the term r is applied at the top position of the result. If one of the applications of the term r leads to a failure, then this failure is propagated and the empty set is obtained as the result of the derivation.

When using a confluent strategy, as the ones presented in *Part I*, the derivation presented above is possible only if the term $G_{sds}(r)$ cannot be reduced to a set with more than one element. This condition is obviously not respected if r is a set with more than one element since, for example, $G_{sds}(\{a,b\}) \xrightarrow{*}_\rho \{G_{sds}(a), G_{sds}(b)\}$. We want to prevent the evaluation of the term $G_{sds}(r)$ to a set with more than one element even when r does not satisfy this condition and therefore, we define the term

$$G_{sd}(r) \triangleq f \to (x \to \langle [\Psi(f);r](x)\rangle)$$

and respectively SD (for *SpreadDown*),

$$SD(r) \triangleq [\Theta](G_{sd}(r)).$$

If $r = \{a,b\}$ then, the term $G_{sd}(r) = G_{sd}(\{a,b\})$ is not reduced to the term $\{G_{sd}(a), G_{sd}(b)\}$ as it was the case for $G_{sds}(r)$ but

$$
\begin{array}{ll}
 & G_{sd}(r) \;\triangleq\; f \to (x \to \langle [\Psi(f);\{a,b\}](x)\rangle) \\
\xrightarrow{*}_\rho & f \to (x \to \langle [\{a,b\}]([\Psi(f)](x))\rangle) \\
\xrightarrow{*}_{Distrib} & f \to (x \to \langle \{[a]([\Psi(f)](x)), [b]([\Psi(f)](x))\}\rangle)
\end{array}
$$

In this last term, the first argument of the operator $\langle\rangle$ contains the free variable x and thus, it cannot be reduced by using the evaluation rule $FirstSuccess$.

Since this last term is not a set, the propagation of the set symbols is not performed in the case of the operator G_{sd} and we can reduce the term $[\Theta](G_{sd}(r))$ to $\{[G_{sd}(r)]([\Theta](G_{sd}(r)))\}$. Consequently, we obtain the reduction:

$$
\begin{array}{ll}
 & [SD(r)](t) \triangleq [[\Theta](G_{sd}(r))](t) \\
\xrightarrow{*}_{\rho} & \{[[G_{sd}(r)]([\Theta](G_{sd}(r)))](t)\} \\
\triangleq & \{[[G_{sd}(r)](SD(r))](t)\} \\
\triangleq & \{[[f \to (x \to \langle[\Psi(f);r](x)\rangle)](SD(r))](t)\} \\
\xrightarrow{*}_{\rho} & \{[\{x \to \langle[\Psi(SD(r));r](x)\rangle\}](t)\} \\
\xrightarrow{*}_{\rho} & \{\langle[\Psi(SD(r));r](f(t_1,\dots,t_n))\rangle\} \\
\xrightarrow{*}_{\rho} & \{\langle[r](f([SD(r)](t_1),\dots,[SD(r)](t_n)))\rangle\}
\end{array}
$$

Example 2.7 If we use a strategy which initially applies the evaluation rules at the top positions of terms then, the following derivation is obtained:

$$
\begin{array}{ll}
 & [SD(\{a \to b, id\})](g(a, f(a))) \\
\xrightarrow{*}_{\rho} & \{\langle[\{a \to b, id\}](g([SD(\{a \to b, id\})](a), [SD(\{a \to b, id\})](f(a))))\rangle\} \\
\longrightarrow_{Distrib} & \{\langle\{[a \to b](g([SD(\{a \to b, id\})](a), [SD(\{a \to b, id\})](f(a)))), \\
 & \quad [id](g([SD(\{a \to b, id\})](a), [SD(\{a \to b, id\})](f(a))))\}\rangle\} \\
\longrightarrow_{Fire} & \{\langle\{\emptyset, [id](g([SD(\{a \to b, id\})](a), [SD(\{a \to b, id\})](f(a))))\}\rangle\} \\
\longrightarrow_{Flat} & \{\langle\{g([SD(\{a \to b, id\})](a), [SD(\{a \to b, id\})](f(a)))\}\rangle\} \\
\xrightarrow{*}_{\rho} & \{\langle\{g(\{\langle[\{a \to b, id\}](a)\rangle\}, [SD(\{a \to b, id\})](f(a)))\}\rangle\} \\
\xrightarrow{*}_{\rho} & \{\langle\{g(\{\{b,a\}, [SD(\{a \to b, id\})](f(a)))\}\rangle\} \\
\xrightarrow{*}_{\rho} & \{\langle\{g(\{b,a\}, \langle[\{a \to b, id\}](f([SD(\{a \to b, id\})](a)))\rangle)\}\rangle\} \\
\xrightarrow{*}_{\rho} & \{\langle\{g(\{b,a\}, f(\{b,a\}))\}\rangle\} \\
\xrightarrow{*}_{\rho} & \{g(b, f(b)), g(a, f(b)), g(b, f(a)), g(a, f(a))\}
\end{array}
$$

We can notice that the application $[SD(r)](t)$ does not guarantee that the applications of the term r to the deepest sub-terms of t are the first ones to be reduced. For example, since we try to apply the evaluation rules at the top position, in the derivation of Example 2.7 we obtain, by applying the evaluation rule $Fire$,

$$[a \to b](g([SD(\{a \to b, id\})](a), [SD(\{a \to b, id\})](f(a)))) \longrightarrow_{Fire} \emptyset$$

and not

$$
\begin{array}{c}
[a \to b](g([SD(\{a \to b, id\})](a), [SD(\{a \to b, id\})](f(a)))) \\
\xrightarrow{*}_{\rho} [a \to b](g(\{b,a\}, \{f(\{b,a\})\})) \xrightarrow{*}_{\rho} \emptyset
\end{array}
$$

as in an *innermost* reduction.

The disadvantage of the non-confluence in the case of the operator SDS was eliminated by using the operator $\langle\rangle$ in the definition of the operator SD, but we have not obtained yet the desired behavior for this type of iterator. In the evaluation of the term $[SD(r)](t)$, if one of the applications of the term r to a sub-term of t is evaluated to $\emptyset$ then, this failure is propagated and the empty set is obtained as the result of the reduction.

If we want to keep unchanged the sub-terms of t on which the application of the term r evaluates to $\emptyset$, we can use the term id either in the same way as in Example 2.7, or by defining the operator G_{bu}:

$$G_{bu}(r) \triangleq f \to (x \to [first(\Psi(f), id); first(r, id)](x))$$

In the same manner as for the previous cases we obtain the operator $BottomUp$:

$$BottomUp(r) \triangleq [\Theta](G_{bu}(r))$$

corresponding to the description presented at the beginning of this section.

Lemma 2.8 The $BottomUp$ operator describing the application of a term to all the sub-terms of another term in a *bottom-up* manner can be expressed in the ρ^{1st}-calculus.

PROOF. We analyze the reductions of the application of a term $BottomUp(r)$ to a constant and to a functional term with several arguments. A complete proof is given in [9]. □

A *top-down* like reduction is immediately obtained if we take the term

$$G_{td}(r) \triangleq f \to (x \to \langle [first(r, id); first(\Psi(f), id)](x) \rangle)$$

and we define the term

$$TopDown(r) \triangleq [\Theta](G_{td}(r)).$$

Lemma 2.9 The $TopDown$ operator describing the application of a term to all the sub-terms of another term in a *top-down* manner can be expressed in the ρ^{1st}-calculus.

2.4.2 Singular applications

Using the term traversal operator Φ we can define similar ρ-terms that apply a specific term only at one position of a ρ-term in a *bottom-up* or *top-down* way. We will see that the operators built using the Φ operator are convenient for the construction of normalization operators.

The ρ-term used in the *bottom-up* case is

$$H_{bu}(r) \triangleq f \to (x \to [first(\Phi(f), r)](x))$$

and we define an operator that applies only once a ρ-term in a *bottom-up* way,

$$Once_{bu}(r) \triangleq [\Theta](H_{bu}(r)).$$

As for the previous operators, the term $[Once_{bu}(r)](t) \triangleq [[\Theta](H_{bu}(r))](t)$ can lead to an infinite reduction if an appropriate strategy is not employed. As for the $SpreadDown$ operator it is enough to apply the evaluation rules first to the top position and only if this is not possible, to deeper positions. We can state:

Lemma 2.10 The $Once_{bu}$ operator describing the application of a term to a sub-term of another term in a *bottom-up* manner can be expressed in the ρ^{1st}-calculus.

Example 2.11 The application $[Once_{bu}(a \to b)](a)$ is reduced to $\{\langle [(a \to b)](a) \rangle\}$ and thus, to the term $\{b\}$.

The application of the rule $a \to b$ to the leftmost-innermost position of a term $g(a, f(a))$ is represented by the term $[Once_{bu}(a \to b)](g(a, f(a)))$ and the corresponding evaluation is presented below:

$$\begin{array}{l} [Once_{bu}(a \to b)](g(a, f(a))) \\ \xrightarrow{*}_{\rho} \{\langle\langle g([Once_{bu}(a \to b)](a), f(a)), g(a, [Once_{bu}(a \to b)](f(a)))\rangle, [a \to b](g(a, f(a)))\rangle\} \\ \xrightarrow{*}_{\rho} \{\langle\langle g(\{b\}, f(a)), g(a, [Once_{bu}(a \to b)](f(a)))\rangle, [a \to b](g(a, f(a)))\rangle\} \\ \xrightarrow{*}_{\rho} \{\langle \{g(b, f(a))\}, [a \to b](g(a, f(a)))\rangle\} \\ \xrightarrow{*}_{\rho} \{g(b, f(a))\} \end{array}$$

If we want to define an operator that applies a specific term only at one position of a ρ-term in a *top-down* way we should use the ρ-term

$$H_{td}(r) \triangleq f \to (x \to [first(r, \Phi(f))](x))$$

and we obtain immediately the operator $Once_{td}$,

$$Once_{td}(r) \triangleq [\Theta](H_{td}(r)).$$

In the case of an application $[Once_{td}(r)](t)$, the application of the term r is first tried at the top position of t and in the case of a failure, r is applied deeper in the term t. As previously, we can state:

Lemma 2.12 The $Once_{td}$ operator describing the application of a term to a sub-term of another term in a *top-down* manner can be expressed in the ρ^{1st}-calculus.

2.5 Repetition and normalization operators

In the previous sections we have defined operators that describe the application of a term at some position of another term (*e.g.* $Once_{bu}$) and operators that allow us to recover from failing evaluations ($first$).

Now we want to define an operator that applies repeatedly a given strategy r to a ρ-term t. We call it *repeat* and its behavior can be described by the following evaluation rule:

$$Repeat \quad [repeat(r)](t) \implies [repeat(r)]([r](t))$$

We use once again the fixed-point operator presented in the previous section and we define the ρ-term

$$I(r) \triangleq f \to (x \to [r; f](x))$$

that is used for describing a *repeat* operator,

$$repeat(r) \triangleq [\Theta](I(r)).$$

This approach has two obvious drawbacks. First, the termination of the evaluation is not guaranteed even when the strategy used for the previous operators is used.

When the strategy applies the evaluation rules first to the top position of an application $[u](v)$ and only afterwards to the right sub-term v and then to the left sub-term u, we do not obtain the desired result. When using this *rightmost-outermost* strategy, the following non-terminating derivation is obtained:

$$[repeat(r)](t) \xrightarrow{*}_\rho \{[repeat(r)]([r](t))\} \xrightarrow{*}_\rho \ldots$$

$$\xrightarrow{*}_\rho \{[repeat(r)]([r]([r](\ldots [r](t) \ldots)))\} \xrightarrow{*}_\rho \ldots$$

Second, when the evaluation terminates the result is always the empty set. If at some point in the evaluation the application of the term r is reduced to the empty set, then $\emptyset$ is strictly propagated and thus the term $[repeat(r)](t)$ is reduced to the empty set.

$Repeat*'$	$[repeat*(r)](t)$	$\Longrightarrow$	$[repeat*(r)]([r](t))$ if $[r](t)$ is not reduced to $\emptyset$
$Repeat*''$	$[repeat*(r)](t)$	$\Longrightarrow$	t if $[r](t)$ is reduced to $\emptyset$

FIG. 3. The operator *repeat∗*

In order to overcome these two problems, we can define an operator called *repeat∗* with a behavior defined by the evaluation rules presented in Figure 3.

Hence, we need an operator similar to the *repeat* one, that stores the last non-failing result and when no further application is possible returns this result. We modify the term $I(r)$ that becomes

$$J(r) \triangleq f \to (x \to [first(r; f,\ id)](x))$$

and we define, as before, the term

$$repeat*(r) \triangleq [\Theta](J(r))$$

We should not forget that we assume here that an application $[u](v)$ is reduced by applying the evaluation rules at the top position, then to its argument v and only afterwards to the term u. Once again, we get:

Lemma 2.13 The operator *repeat∗* describing the repeated application of a term while the result is not $\emptyset$ can be expressed in the ρ^{1st}-calculus.

Example 2.14 The repeated application of the rewrite rules $a \to b$ and $b \to c$ on the term a is represented by the term $[repeat*(\{a \to b, b \to c\})](a)$ that evaluates as follows:

$$
\begin{array}{l}
\quad [repeat*(\{a \to b, b \to c\})](a) \\
\xrightarrow{*}_{\rho} \{\langle [repeat*(\{a \to b, b \to c\})]([\{a \to b, b \to c\}](a)), [id](a)\rangle\} \\
\xrightarrow{*}_{\rho} \{\langle [repeat*(\{a \to b, b \to c\})](\{b\}), [id](a)\rangle\} \\
\xrightarrow{*}_{\rho} \{\langle\{\langle [repeat*(\{a \to b, b \to c\})]([\{a \to b, b \to c\}](b)), [id](b)\rangle\}, [id](a)\rangle\} \\
\xrightarrow{*}_{\rho} \{\langle\{\langle [repeat*(\{a \to b, b \to c\})](\{c\}), [id](b)\rangle\}, [id](a)\rangle\} \\
\xrightarrow{*}_{\rho} \{\langle\{\langle\{\langle [repeat*(\{a \to b, b \to c\})]([\{a \to b, b \to c\}](c)), [id](c)\rangle\}, [id](b)\rangle\}, [id](a)\rangle\} \\
\xrightarrow{*}_{\rho} \{\langle\{\langle\{\langle [repeat*(\{a \to b, b \to c\})](\emptyset), \{c\}\rangle\}, [id](b)\rangle\}, [id](a)\rangle\} \\
\xrightarrow{*}_{\rho} \{\langle\{\langle\{\langle \emptyset, \{c\}\rangle\}, [id](b)\rangle\}, [id](a)\rangle\} \\
\xrightarrow{*}_{\rho} \{\langle\{\langle \{c\}, [id](b)\rangle\}, [id](a)\rangle\} \\
\xrightarrow{*}_{\rho} \{\langle \{\{c\}\}, [id](a)\rangle\} \\
\xrightarrow{*}_{\rho} \{c\}
\end{array}
$$

Using the above operators it is easy to define some specific normalization strategies. For example, the *innermost* strategy is defined by

$$im(r) \triangleq repeat*(Once_{bu}(r))$$

and an *outermost* strategy is defined by

$$om(r) \triangleq repeat{*}(Once_{td}(r)).$$

Corollary 2.15 The operators im et om describing the *innermost* and *outermost* normalization can be expressed in the ρ^{1st}-calculus.

We have now all the ingredients needed for describing the normalization of a term t in a rewrite theory $\mathcal{R}$. The term $\xi_{\mathcal{R}}(u)$ described at the beginning of this section can be defined using the $im(\mathcal{R})$ or $om(\mathcal{R})$ operators and thus, we can represent the normalization of a term u w.r.t. a rewriting theory $\mathcal{R}$ by the ρ-terms

$$\xi_{\mathcal{R}}(u) \triangleq [im(\mathcal{R})](u)$$

or

$$\xi_{\mathcal{R}}(u) \triangleq [om(\mathcal{R})](u).$$

Example 2.16 If we denote by $\mathcal{R}$ the set of rewrite rules $\{a \to b, g(x, f(x)) \to x\}$, we represent by $[im(\mathcal{R})](g(a, f(a)))$ the leftmost-innermost normalization of the term $g(a, f(a))$ according to the set of rules $\mathcal{R}$ and the following derivation is obtained:

$$\begin{array}{ll}
 & [im(\mathcal{R})](g(a,f(a))) \\
\triangleq & [repeat{*}(Once_{bu}(\mathcal{R}))](g(a,f(a))) \\
\xrightarrow{*}_{\rho} & \{\langle [repeat{*}(Once_{bu}(\mathcal{R}))]([Once_{bu}(\mathcal{R})](g(a,f(a)))), [id](g(a,f(a)))\rangle\} \\
\xrightarrow{*}_{\rho} & \{\langle [repeat{*}(Once_{bu}(\mathcal{R}))](\{g(b,f(a))\}), [id](g(a,f(a)))\rangle\} \\
\xrightarrow{*}_{\rho} & \{\langle\{[repeat{*}(Once_{bu}(\mathcal{R}))](g(b,f(a)))\}, [id](g(a,f(a)))\rangle\} \\
\xrightarrow{*}_{\rho} & \{\langle\{\{\langle [repeat{*}(Once_{bu}(\mathcal{R}))]([Once_{bu}(\mathcal{R})](g(b,f(a)))), \\
 & \quad [id](g(b,f(a)))\rangle\}\}, [id](g(a,f(a)))\rangle\} \\
\xrightarrow{*}_{\rho} & \{\langle\{\langle [repeat{*}(Once_{bu}(\mathcal{R}))](\{g(b,f(b))\}), \\
 & \quad [id](g(b,f(a)))\rangle\}, [id](g(a,f(a)))\rangle\} \\
\xrightarrow{*}_{\rho} & \{\langle\{\langle\{\langle\{[repeat{*}(Once_{bu}(\mathcal{R}))]([Once_{bu}(\mathcal{R})](g(b,f(b))))\}, \\
 & \quad [id](g(b,f(b)))\rangle\}, [id](g(b,f(a)))\rangle\}, [id](g(a,f(a)))\rangle\} \\
\xrightarrow{*}_{\rho} & \{\langle\{\langle\{\langle\{[repeat{*}(Once_{bu}(\mathcal{R}))](\{b\})\}, \\
 & \quad [id](g(b,f(b)))\rangle\}, [id](g(b,f(a)))\rangle\}, [id](g(a,f(a)))\rangle\} \\
\xrightarrow{*}_{\rho} & \{\langle\{\langle\{\langle\{\{\langle [repeat{*}(Once_{bu}(\mathcal{R}))]([Once_{bu}(\mathcal{R})](b)), [id](b)\rangle, \\
 & \quad [id](g(b,f(b)))\rangle\}, [id](g(b,f(a)))\rangle\}, [id](g(a,f(a)))\rangle\} \\
\xrightarrow{*}_{\rho} & \{\langle\{\langle\{\langle\{\{\langle [repeat{*}(Once_{bu}(\mathcal{R}))](\emptyset), [id](b)\rangle, \\
 & \quad [id](g(b,f(b)))\rangle\}, [id](g(b,f(a)))\rangle\}, [id](g(a,f(a)))\rangle\} \\
\xrightarrow{*}_{\rho} & \{\langle\{\langle\{\langle\{\{\langle \emptyset, [id](b)\rangle, \\
 & \quad [id](g(b,f(b)))\rangle\}, [id](g(b,f(a)))\rangle\}, [id](g(a,f(a)))\rangle\} \\
\xrightarrow{*}_{\rho} & \{\langle\{\langle\{\langle\{\{\{b\}\}\}, [id](g(b,f(b)))\rangle\}, [id](g(b,f(a)))\rangle\}, [id](g(a,f(a)))\rangle\} \\
\xrightarrow{*}_{\rho} & \{\langle\{\langle\{\{b\}\}, [id](g(b,f(a)))\rangle\}, [id](g(a,f(a)))\rangle\} \\
\xrightarrow{*}_{\rho} & \{\langle\{\{b\}\}, [id](g(a,f(a)))\rangle\} \\
\xrightarrow{*}_{\rho} & \{\langle\{b\}, [id](g(a,f(a)))\rangle\} \\
\xrightarrow{*}_{\rho} & \{b\}
\end{array}$$

Given a term u, if the rewriting theory $\mathcal{R}$ is not confluent then, the result of the reduction of the term $[im(\mathcal{R})](u)$ is a set representing all the possible results of the

reduction of the term u in the rewriting theory $\mathcal{R}$. Each of the elements of the result set represents the result of a reduction in the rewriting theory $\mathcal{R}$ for a given application order of the rewrite rules in $\mathcal{R}$.

Example 2.17 Let us consider the set $\mathcal{R} = \{a \to b, a \to c, g(x,x) \to x\}$ of non-confluent rewrite rules. The term $[im(\mathcal{R})](g(a,a))$ representing the *innermost* normalization of the term $g(a,a)$ according to the set of rewrite rules $\mathcal{R}$ is reduced to $\{b, g(c,b), g(b,c), c\}$. The term $[om(\mathcal{R})](g(a,a))$ representing the *outermost* normalization is reduced to $\{b, c\}$.

We have now all the ingredients necessary to describe in a concise way the normalization process induced by a rewrite theory. Of course, the standard properties of termination and confluence of the rewrite system will allow us to get uniqueness of the result. Our approach differs from this and we define this normalization even in the case where there is no unique normal form or where termination is not warranted. This is why in general we do not get termination or uniqueness of the normal form.

3 Using the ρ^{1st}-calculus

We have shown in *Part I* that a finite derivation in term rewriting can be mimicked as an appropriate ρ-term that indeed represents the trace of the reduction. It is often more interesting to *find* such a derivation.

3.1 Encoding rewriting in the ρ^{1st}-calculus

We are interested to build a ρ-term describing the reduction, in term rewriting, of term t w.r.t. a set of rewrite rules, but without knowing *a priori* the intermediate steps of the derivation of t. For this, we can use the ρ_T^{1st}-calculus and the operators defining *innermost* and *outermost* normalization strategies.

Proposition 3.1 Given a rewriting theory $\mathcal{T}_\mathcal{R}$ and two first order ground terms $t, t{\downarrow} \in \mathcal{T}(\mathcal{F})$ such that t is normalized to $t{\downarrow}$ w.r.t. the set of rewrite rules $\mathcal{R}$. Then, $[im(\mathcal{R})](t)$ is ρ-reduced to a set containing the term $t{\downarrow}$.

PROOF. By induction on the number of reduction steps for the term t.

□

Example 3.2 Let us consider a rewrite system $\mathcal{R}$ containing the rewrite rules $(x = x) \to True$ and $b \to a$. Then, the term $a = b$ reduces to $True$ in this rewrite system and a ρ-term reducing to $\{True\}$ can be built as shown in *Part I* or using the fixed-point operators.

In the former case the corresponding ρ-term is

$$[(x = x) \to True]([a = (b \to a)](a = b)).$$

For the latter approach we build the term

$$[im(\{(x = x) \to True, b \to a\})](a = b).$$

Since in this case we can obtain empty sets and additionally, sets with more than one element are obtained when equational matching is not unitary, a reduction strategy

as presented in *Part I* should be used in order to ensure confluence. If no reduction strategy is used then undesired results can be obtained.

3.2 Encoding conditional rewriting

As shown before, any term rewriting reduction can be described by a reduction in the ρ-calculus. In this section we give a representation in the ρ-calculus of the conditional rewriting reductions. We will propose thus, methods for defining a ρ-term that contains all the information needed for reduction including the condition evaluation that is normally performed on the meta-level.

The main difficulty here resides in the fact that for conditional rewriting, the reduction relation is recursively applied in order to evaluate the condition when firing a conditional rule. We can use the same approach as our explicit description of non-conditional rewriting (see *Part I*) but the ρ-terms used in order to describe the conditional rewriting reduction become very complicated in this case. Instead, a detailed description by a concise ρ-term of the normalization process of the conditions can be obtained by using the normalization operators presented in the Section 2.5.

3.2.1 Definition of conditional rewriting

Many conditional rewriting relations have been designed and mainly differ in the way the conditions are understood [15]. We consider here the normal conditional rewriting defined as follows.

Definition 3.3 A *normal* rewrite system $\mathcal{R}$ is composed of conditional rewrite rules of the form $(l \to r \text{ if } c)$ where l, r, c are elements of $\mathcal{T}(\mathcal{F}, \mathcal{X})$ with variables satisfying the condition $\mathcal{V}ar(r) \cup \mathcal{V}ar(c) \subseteq \mathcal{V}ar(l)$, and such that for each ground substitution σ satisfying $\mathcal{V}ar(c) \subseteq \mathcal{D}om(\sigma)$, the normal form under $\mathcal{R}$ of σc is either the boolean $True$ or $False$. Given a conditional rewrite system $\mathcal{R}$ composed of such rules, the application of the rewrite rule $(l \to r \text{ if } c)$ of $\mathcal{R}$ on a term t at occurrence m consists in:

(i) matching, using the substitution σ, the left-hand side of the rule against the term $t_{|m}$

(ii) normalizing the instantiated condition σc using $\mathcal{R}$ and, provided the resulting term is $True$,

(iii) replace $t_{|m}$ by σr in t.

This is denoted $t \longrightarrow_{[m]}^{l \to r \text{ if } c} t_{\lceil \sigma r \rceil_m}$.

3.2.2 Encoding

As we have mentioned, the main difficulty in the encoding of conditional rewriting is to make precise the evaluation process of the condition. In the case of normal rewriting, this means computing the normal form of the condition.

We denote by c_ρ the ρ-term that, when instantiated by the proper substitution (*i.e.* θc_ρ), normalizes to the term $\{u\}$ if the term c, instantiated accordingly (*i.e.* θc), is normalized into u in the rewrite theory $\mathcal{R}$. When the term c is a boolean condition

and when the rewrite system is completely defined over the booleans [5], the term u should be one of the two constants $True$ or $False$.

If the reduction in a rewrite theory $\mathcal{R}$ is known, we can define, as in *Part I*, the ρ-term $c_\rho \triangleq [u_n](\dots[u_1](c)\dots)$ that evaluates to $\{u\}$, *i.e.* to $\{True\}$ or $\{False\}$. If c_ρ is the ρ-term describing the reduction of the term c then, the conditional rewrite rule $l \to r$ if c is represented by the ρ-term

$$l \to [\{True \to r, False \to \emptyset\}](c_\rho)$$

or even the simpler, but maybe less suggestive one,

$$l \to [True \to r](c_\rho).$$

In the case when c_ρ reduces to $\{False\}$, in the latter representation the matching fails and the result of the application is, as in the former one, the empty set. When c_ρ reduces to $\{True\}$, the result of the reduction is obviously the same in the two cases, *i.e.* the same as the application of $l \to r$.

By using the above representation, we can extend the Proposition given in *Part I* and show that any derivation in a conditional rewriting theory is representable by an appropriate ρ-term.

Proposition 3.4 Given a conditional rewriting theory $\mathcal{T}_\mathcal{R}$ and two first order ground terms $t, t' \in \mathcal{T}(\mathcal{F})$ such that $t \xrightarrow{*}_\mathcal{R} t'$. Then, there exist the ρ-terms $u_1, \dots, u_n$ built using the rewrite rules in $\mathcal{R}$ and the intermediate steps in the derivation $t \xrightarrow{*}_\mathcal{R} t'$ such that we have $[u_n](\dots[u_1](t)\dots) \xrightarrow{*}_{\rho_\emptyset} \{t'\}$.

The construction approach used in *Part I* for unconditional rewriting is obviously not convenient and we need a method that allows us to build the ρ-term corresponding to a rewrite reduction without knowing *a priori* the reduction steps. In order to build the ρ-term c_ρ using only the term c and the rewrite rules of $\mathcal{R}$, we can use the normalization operators defined in Section 2. For example, we can define

$$c_\rho \triangleq [im(\mathcal{R})](c).$$

Example 3.5 Let us assume that the set of rules describing the order on integers is denoted by $\mathcal{R}_<$. We consider the rewrite rule ($f(x) \to g(x)$ if $x \geq 1$) that applied to the term $f(2)$ reduces to $g(2)$ since x is instantiated by 2 and the condition $(2 \geq 1)$ reduces to $True$ by using the rewrite rule $(2 \geq 1) \to True$.

If we consider that the condition is normalized according to $\mathcal{R}_<$, then the corresponding reduction in the ρ-calculus is the following:

$$\begin{array}{ll} & [f(x) \to [True \to g(x)]([im(\mathcal{R}_<)](x \geq 1))](f(2)) \\ \longrightarrow_{Fire} & \{[True \to g(2)]([im(\mathcal{R}_<)](2 \geq 1))\} \\ \xrightarrow{*}_{\rho} & \{[True \to g(2)](\{True\})\} \\ \longrightarrow_{Batch} & \{\{[True \to g(2)](True)\}\} \\ \longrightarrow_{Fire} & \{\{\{g(2)\}\}\} \\ \xrightarrow{*}_{Flat} & \{g(2)\} \end{array}$$

The conditions of the rewrite rules can be normalized according to a set of conditional rewrite rules, including the current rule, and thus the definition of the ρ-rewrite

rules representing this normalization is intrinsically recursive and cannot be realized only by using the operator im.

We use the fixed-point operator Θ described in Section 2.4 to represent the application of the same set of rewrite rules for the normalization of all the conditions.

Given a set of rewrite rules $\mathcal{R} = \mathcal{R}_n \cup \mathcal{R}_c$ where $\mathcal{R}_n$ and $\mathcal{R}_c$ represent the subset of non-conditional rewrite rules and respectively the subset of conditional rewrite rules of the form $(l \to r \text{ if } c)$. We define the term

$$R \triangleq f \to (y \to [im(\{l_i \to [True \to r_i]([f](c_i)) \mid i = 1 \ldots m\} \cup \mathcal{R}_n)](y))$$

where $\mathcal{R}_c = \{l_i \to r_i \text{ if } c_i \mid i = 1 \ldots m\}$, $\mathcal{R}_n = \{l'_i \to r'_i \mid i = 1 \ldots n\}$ and respectively

$$IM(R) \triangleq [\Theta](R).$$

Thus, for describing the normalization of the term t w.r.t. the rewrite rules of $\mathcal{R}$ we use the ρ-term $[IM(R)](t)$.

The normalization strategy for the conditions is now abstracted by the variable f and since $IM(R) \triangleq [\Theta](R)$ is reduced to$[R]([\Theta](R))$ then this variable is instantiated at the beginning by $[\Theta](R)$ (*i.e.* $IM(R)$). Thus, not only the initial term but also the conditions are reduced according to $IM(R)$. This instantiation can be possibly reiterated if some conditional rules suppose the application of other conditional rules.

We obtain thus a result similar to Proposition 3.4 but with a method of construction for the corresponding ρ-term based only on the initial term and on the set of rewrite rules.

Proposition 3.6 Given a conditional rewriting theory $\mathcal{T}_{\mathcal{R}}$ and two first order ground terms $t, t\!\downarrow \in \mathcal{T}(\mathcal{F})$ such that t is normalized to $t\!\downarrow$ w.r.t. the set of rewrite rules $\mathcal{R}$. Then, $[IM(\mathcal{R})](t)$ is ρ-reduced to a set containing the term $t\!\downarrow$.

Example 3.7 We consider the set of rewrite rules $\mathcal{R}$ containing the rewrite rule $(x = x) \to True$ and the conditional rewrite rules $(f(x) \to g(x) \text{ if } h(x) = b)$ and $(h(x) \to b \text{ if } x = a)$. The term $f(a)$ reduces to $g(a)$ using the rewrite rules of $\mathcal{R}$ and we show below the corresponding reduction in ρ-calculus.

Using the method presented above we obtain the ρ-term:

$$\begin{aligned} R \triangleq f \to (y \to [im(\{&f(x) \to [True \to g(x)]([f](h(x) = b)),\\ &h(x) \to [True \to b]([f](x = a)),\\ &(x = x) \to True\\ \})](y))& \end{aligned}$$

We show the main steps in the reduction of the term $[IM(R)](f(a))$. We obtain immediately the reduction

$$[IM(R)](f(a)) \triangleq [[\Theta](R)](f(a)) \quad \xrightarrow{*}_{\rho} \quad [[R]([\Theta](R))](f(a)) \triangleq [[R](IM(R))](f(a))$$

and the final result is the same as the one obtained for the term

$$\begin{aligned} [im(\{&f(x) \to [True \to g(x)]([IM(R)](h(x) = b)),\\ &h(x) \to [True \to b]([IM(R)](x = a)),\\ &(x = x) \to True\\ \})](&f(a))) \end{aligned}$$

and thus for

$$[f(x) \to [True \to g(x)]([IM(R)](h(x)=b))](f(a))$$
$$\stackrel{*}{\longrightarrow}_\rho \quad \{[True \to g(a)]([IM(R)](h(a)=b))\}$$

For the term $[IM(R)](h(a)=b)$ we proceed as previously and thus, we have to reduce the term

$$\begin{array}{l} [im(\{f(x) \to [True \to g(x)]([IM(R)](h(x)=b)), \\ \quad h(x) \to [True \to b]([IM(R)](x=a)), \\ \quad (x=x) \to True\} \\ )](h(a)=b) \end{array}$$

with the intermediate reduction

$$[h(x) \to [True \to b]([IM(R)](x=a))](h(a)) \quad \stackrel{*}{\longrightarrow}_\rho \quad \{[True \to b]([IM(R)](a=a))\}$$

Since we easily obtain $[IM(R)](a=a) \stackrel{*}{\longrightarrow}_\rho \{True\}$ then, the previous term is reduced to $\{[True \to b](\{True\})\} \stackrel{*}{\longrightarrow}_\rho \{b\}$ and we have

$$[IM(R)](h(a)=b) \quad \stackrel{*}{\longrightarrow}_\rho \quad [im(\dots)](\{b\}=b) \quad \stackrel{*}{\longrightarrow}_\rho \quad \{True\}$$

We come back to the reduction of the initial term and we get

$$\{[True \to g(a)]([IM(R)](h(a)=b))\} \quad \stackrel{*}{\longrightarrow}_\rho \quad \{[True \to g(a)](\{True\})\} \quad \stackrel{*}{\longrightarrow}_\rho \quad \{g(a)\}$$

We have thus obtained the same result as in conditional term rewriting.

Starting from the results presented in this section we will give in the next section a representation of the more elaborated rewrite rules used in ELAN, a language based on conditional rewrite rules with local assignments.

4 The rewriting calculus as a semantics of ELAN

4.1 ELAN*'s rewrite rules*

ELAN (a name that expresses the dynamism of the arrow), is an environment for specifying and prototyping deduction systems in a language based on labeled conditional rewrite rules and strategies to control rule application. The ELAN system offers a compiler and an interpreter of the language. The ELAN language allows us to describe in a natural and elegant way various deduction systems [29, 19, 3]. It has been experimented on several non-trivial applications ranging from decision procedures, constraint solvers [6], logic programming [21] and automated theorem proving [10] but also specification and exhaustive verification of authentication protocols [8].

ELAN's rewrite rules are conditional rewrite rules with local assignments. The local assignments are let-like constructions that allow applications of strategies to some terms. The general syntax of an ELAN rule is:

$$[\ell] \quad l \Rightarrow r \qquad [\ \textbf{if}\ \ cond \quad | \quad \textbf{where}\ \ y := (S)u\ ]^* \quad end$$

where *cond* is an ELAN expression that can be reduced to a boolean value. If all the conditions are reduced to the `true` value and all local variables (*e.g.* y) are assigned

with success (*i.e.* the application of the strategy from the right-hand side of the local assignment does not fail) then the rewrite rule can be applied.

We should notice that the square brackets ([ ]) in ELAN are used to indicate the label of the rule and should be distinguished from the square brackets of the ρ-calculus that represent the application of a rewrite rule (ρ-term).

A partial semantics could be given to an ELAN program using rewriting logic [22, 4], but more conveniently all ELAN's rules (and not only the conditional ones) and strategies can be expressed using the ρ-calculus and thus an ELAN program is just a a ρ-term. The results of the evaluation of this ρ-term correspond to all the possible results of the execution of the initial ELAN program.

Example 4.1 An example of a labeled ELAN rule describing a possible naive way to search the minimal element of a list by sorting the list and taking the first element is the following:

```
[min-rule]  min(l)   =>   m
                 if l !=  nil
                 where sl := (sort) l
                 where m  := () head(sl)  end
```

The strategy `sort` can be any sorting strategy. The operator `head` is supposed to be described by a confluent and terminating set of unlabeled rewrite rules. Thus, `sl` is assigned the result of the application of a given set of labeled rules guided by the strategy `(sort)`, while `m` is assigned the result of the application of a given set of unlabeled rules guided by the strategy `()` (*i.e.* the implicit built-in *innermost* strategy).

The evaluation strategy used for evaluating the conditions is a leftmost innermost standard rewriting strategy.

The non-determinism is handled mainly by two basic strategy operators: `dont care choose` (denoted `dc`$(s_1, \ldots, s_n)$) that returns the results of at most one non-deterministicly chosen unfailing strategy from its arguments and `dont know choose` (denoted `dk`$(s_1, \ldots, s_n)$) that returns all the possible results. A variant of the `dont care choose` operator is the `first choose` operator (denoted `first`$(s_1, \ldots, s_n)$) that returns the results of the first unfailing strategy from its arguments.

Several strategy operators implemented in ELAN allow us a simple and concise description of user defined strategies. For example, the concatenation operator denoted "`;`" builds the sequential composition of two strategies s_1 and s_2. The strategy $s_1; s_2$ fails if s_1 fails, otherwise it returns all results (maybe none) of s_2 applied to the results of s_1. Using the operator `repeat*` we can describe the repeated application of a given strategy. Thus, `repeat*`(s) iterates the strategy s until it fails and then returns the last obtained result.

Any rule in ELAN is considered as a basic strategy and several other strategy operators are available for describing the computations. Here is a simple example illustrating the way the `first` and `dk` strategies work.

Example 4.2 If the strategy `dk(x=>x+1,x=>x+2)` is applied to the term a, ELAN provides two results: $a+1$ and $a+2$. When the strategy `first(x=>x+1,x=>x+2)` is applied to the same term only the $a+1$ result is obtained. The strategy `first(b=>b+1, a=>a+2)` applied to the term a yields the result $a+2$.

Using non-deterministic strategies, we can explore exhaustively the search space of a given problem and find paths described by some specific properties.

For example, for proving the correctness of the Needham-Schroeder authentication protocol [24] we look for possible attacks among all the behaviors during a session. In Example 4.3 we present just one of the rules of the protocol and we give the strategy looking for all the possible attacks, a more detailed description of the implementation is given in [8].

Example 4.3 We consider the rewrite rules describing the Needham-Schroeder authentication protocol that aims to establish a mutual authentication between an initiator and a responder that communicate via an insecure network (*i.e.* in presence of intruders).

The strategy looking for possible attacks applies repeatedly and non-deterministicly all the rewrite rules describing the behavior of the protocol (*e.g.* initiate) and of the intruder (*e.g.* intruder) and selects only those results representing an attack.

```
[]attStrat => repeat*(
                          dk( initiate, ..., intruder)
              );
              attackFound                                  end
```

The non-deterministic application is described with the operator `dk`. The result of the strategy `repeat*(...)` is the set of all possible behaviors in a protocol session where messages can be intercepted or faked by an intruder. The strategy `attackFound` just checks if the term received as input represents an attack (by trying to apply the rewrite rules corresponding to the negation of the desired invariants) and therefore selects from the previous set of results only those representing an attack.

4.2 The ρ-calculus representation of ELAN *rules*

The rules of the system ELAN can be expressed using the ρ-calculus. A rule with no conditions and no local assignments $l \Rightarrow r$ is represented by $l \to r$ and a conditional rule is expressed as in Section 3.2.

4.2.1 Rules with local assignments

The ELAN rewrite rules with local assignments but without conditions of the form

$$[\ell] \quad l(x) \Rightarrow \quad r(x,y) \quad \texttt{where } y := (S)u$$

can be represented by the ρ-term

$$l(x) \to r(x, [S_\rho](u))$$

or the ρ-term

$$l(x) \to [y \to r(x,y)]([S_\rho](u))$$

with S_ρ, the ρ-term corresponding to the strategy S in the ρ-calculus.

The first representation syntactically replaces all variables of the right-hand side of the rewrite rule defined in a local assignment with the term which instantiates

the respective variable. In the second representation, each variable defined in a local assignment is bound in a ρ-rewrite rule which is applied to the corresponding term.

Example 4.4 The ELAN rule

```
[deriveSum] p_1 + p_2  =>  p_1' + p_2'
                            where p_1' := (derive)p_1
                            where p_2' := (derive)p_2       end
```

can be represented by one of the following two ρ-terms
$p_1 + p_2 \to [derive](p_1) + [derive](p_2)$,
$p_1 + p_2 \to [p_1' \to [p_2' \to p_1' + p_2']([derive](p_2))]([derive](p_1))$.

At this moment one can notice the usefulness of free variables in the rewrite rules. The latter representation of an ELAN rule with local assignments would not be possible if the variable p_1' was not allowed to be free in the ρ-rule $p_2' \to p_1' + p_2'$. The free variables in the right-hand side of a ρ-rewrite-rule also enables the parameterization of rewrite rules by strategies as in $y \to [f(x) \to [y](x)](f(a))$ where the strategy to be applied on x is not known in the rule $f(x) \to [y](x)$.

Example 4.5 We consider the ELAN rule

```
[deriveSum] x  =>  y + y
                    where y := (derive)x       end
```

Let us consider that the strategy `derive` is `dk(a=>b,a=>c)`. Then, the application of the strategy `derive` to the term a gives the two results b and c. Thus, the application of the rule `deriveSum` to the term a provides non-deterministically one of the four results $b + b$, $b + c$, $c + b$, $c + c$.

The ρ-representation of this rule is

$$x \to [\{a \to b, a \to c\}](x) + [\{a \to b, a \to c\}](x)$$

that applied to a reduces as follows

$$\begin{array}{ll}
 & [x \to [\{a \to b, a \to c\}](x) + [\{a \to b, a \to c\}](x)](a) \\
\longrightarrow_{Fire} & \{[\{a \to b, a \to c\}](a) + [\{a \to b, a \to c\}](a)\} \\
\stackrel{*}{\longrightarrow}_{Distrib} & \{\{[a \to b](a), [a \to c](a)\} + \{[a \to b](a), [a \to c](a)\}\} \\
\stackrel{*}{\longrightarrow}_{Fire} & \{\{\{b\}, \{c\}\} + \{\{b\}, \{c\}\}\} \\
\longrightarrow_{Flat} & \{\{b, c\} + \{b, c\}\} \\
\longrightarrow_{OpOnSet} & \{\{b + \{b, c\}, c + \{b, c\}\}\} \\
\longrightarrow_{OpOnSet} & \{\{\{b + b, b + c\}, \{c + b, c + c\}\}\} \\
\longrightarrow_{OpOnSet} & \{\{\{b + b, b + c\}, \{c + b, c + c\}\}\} \\
\stackrel{*}{\longrightarrow}_{Flat} & \{b + b, b + c, c + b, c + c\}
\end{array}$$

This set represents exactly the four results obtained in ELAN.

If we consider more general ELAN rules containing local assignments as well as conditions on the local variables, the combination of the methods used for conditional rules and rules with local assignments should be done carefully. If we had used a representation closed to the first one from Example 4.4 we would have obtained some incorrect results as in Example 4.6.

Example 4.6 We consider the description of an automaton by a set of rewrite rules, each one describing the transition from a state to another. The potential execution of a double transition from an initial state in a final state passing by a non-final intermediate state, can be described by the following ELAN rule:

```
[double] x => next(y)
                    where y := (dk(s1 => s2,s1 => s3)) x
                    if nf(y)
end
```

The term `next(y)` represents the state obtained by carrying out a transition from `y` and this behavior can be easily represented in ELAN by a set of unlabeled rules describing the operator `nf`. We note by $\mathcal{R}_f$ the set of rewrite rules describing the final states and we suppose that `s2` is a final state but `s3` is not.

By using the first representation approach of a rule with local assignments and the coding method for conditional rules presented in Section 3.2, we obtain the ρ-term corresponding to the previous ELAN rule:

$$x \to [True \to next([\{s1 \to s2, s1 \to s3\}](x))]([im(\mathcal{R}_f)](nf([\{s1 \to s2, s1 \to s3\}](x))))$$

This term applied to $s1$ leads to the following reduction

$$
\begin{array}{l}
[x \to [True \to next([\{s1 \to s2, s1 \to s3\}](x))] \\
\qquad ([im(\mathcal{R}_f)](nf([\{s1 \to s2, s1 \to s3\}](x))))](s1) \\
\longrightarrow_\rho \{[True \to next([\{s1 \to s2, s1 \to s3\}](s1))] \\
\qquad ([im(\mathcal{R}_f)](nf([\{s1 \to s2, s1 \to s3\}](s1))))\} \\
\stackrel{*}{\longrightarrow}_\rho \{[True \to next(\{s2, s3\})]([im(\mathcal{R}_f)](nf(\{s2, s3\})))\} \\
\stackrel{*}{\longrightarrow}_\rho \{[True \to \{next(s2), next(s3)\}]([im(\mathcal{R}_f)](\{nf(s2), nf(s3)\}))\} \\
\stackrel{*}{\longrightarrow}_\rho \{[True \to \{next(s2), next(s3)\}](\{False, True\})\} \\
\stackrel{*}{\longrightarrow}_\rho \{\{[True \to \{next(s2), next(s3)\}](False), [True \to \{next(s2), next(s3)\}](True)\}\} \\
\stackrel{*}{\longrightarrow}_\rho \{\emptyset, [True \to \{next(s2), next(s3)\}](True)\} \\
\stackrel{*}{\longrightarrow}_\rho \{\emptyset, \{next(s2), next(s3)\}\} \\
\stackrel{*}{\longrightarrow}_\rho \{next(s2), next(s3)\}
\end{array}
$$

while in ELAN we obtain the only result `next(s3)` that would be represented by the ρ-term $\{next(s3)\}$.

The problem in the Example 4.6 is the double evaluation of the term $[\{s1 \to s2, s1 \to s3\}](s1)$ replacing the local variable y: once in the condition and once in the right-hand side of the rule. If this term is evaluated to a set with more than one element and one of its elements satisfies the condition, then this set replaces the corresponding variables in the right-hand side of the rule, while only the subset of elements satisfying the condition should be considered. Therefore, we need a mechanism that evaluates only once each of the local assignments of a rule.

We use an approach combining the second representation approach of a rule with local assignments and the ρ-representation of conditional rules. Without losing generality, we consider that an ELAN rule that has the following form:

$[label] \quad l \Longrightarrow r_{\lceil x \rceil_q}$

    where $x := (s)t$
    if $C_{\lceil x \rceil_p}$

end

Then, the ELAN rule presented above is expressed as the ρ-term

$$l \to [x \to [\{True \to r_{\lceil x \rceil_q}, False \to \emptyset\}]([im(\mathcal{R})](C_{\lceil x \rceil_p}))]([s](t))$$

or the simpler one

$$l \to [x \to [True \to r_{\lceil x \rceil_q}]([im(\mathcal{R})](C_{\lceil x \rceil_p}))]([s](t))$$

where $\mathcal{R}$ represents the set of rewrite rules modulo which we normalize the conditions.

In order to simplify the presentation we supposed that the rules of the set $\mathcal{R}$ are rewrite rules of the form $l \to r$ and thus, the operator im is sufficient to define normalization w.r.t. such a set. If we consider conditional unlabeled rules, then the operator IM must be employed.

The way the transformation is applied to an ELAN rewrite rule and the corresponding reduction are illustrated by taking again the Example 4.6 and considering the new representation.

Example 4.7 The ELAN rewrite rule from Example 4.6 is represented by the ρ-term

$$x \to [y \to [True \to next(y)]([im(\mathcal{R}_f)](nf(y)))]([\{s1 \to s2, s1 \to s3\}](x))$$

that, applied to the term $s1$ leads to the following reduction

$$\begin{array}{ll}
[x \to [y \to [True \to next(y)]([im(\mathcal{R}_f)](nf(y)))]([\{s1 \to s2, s1 \to s3\}](x))](s1) & \\
\longrightarrow_{Fire} \{[y \to [True \to next(y)]([im(\mathcal{R}_f)](nf(y)))]([\{s1 \to s2, s1 \to s3\}](s1))\} & \\
\stackrel{*}{\longrightarrow}_\rho \quad \{[y \to [True \to next(y)]([im(\mathcal{R}_f)](nf(y)))](\{s2, s3\})\} & \\
\stackrel{*}{\longrightarrow}_\rho \quad \{[y \to [True \to next(y)]([im(\mathcal{R}_f)](nf(y)))](s2), & \\
\qquad [y \to [True \to next(y)]([im(\mathcal{R}_f)](nf(y)))](s3)\} & \\
\stackrel{*}{\longrightarrow}_{Fire} \{\{[True \to next(s2)]([im(\mathcal{R}_f)](nf(s2)))\}, & \\
\qquad \{[True \to next(s3)]([im(\mathcal{R}_f)](nf(s3)))\}\} & \\
\stackrel{*}{\longrightarrow}_\rho \quad \{[True \to next(s2)](False), [True \to next(s3)](True)\} & \\
\stackrel{*}{\longrightarrow}_\rho \quad \{\emptyset, \{next(s3)\}\} & \\
\stackrel{*}{\longrightarrow}_\rho \quad \{next(s3)\} &
\end{array}$$

that is the representation of the result obtained in ELAN.

The same result as in Example 4.6 is obtained if the evaluation rule $Fire$ is applied before the distribution of the set $\{s2, s3\}$. But the confluent strategies presented in *Part I* forbid such a reduction and thus, the correct result is obtained.

This latter representation not only allows a correct transformation of ELAN reductions in ρ-reductions but gives also a hint on the implementation details of such rewrite rules. On one hand the implementation should ensure the correctness of the result and on the other hand it should take into account the efficiency problems. For instance, the representation used in Example 4.5 is correct but obviously less efficient

than a representation as in Example 4.7 and this is due to the double evaluation of the same application.

The ELAN evaluation mechanism is more complex than presented above since it distinguishes between labeled rewrite rules and unlabeled rewrite rules. The unlabeled rewrite rules are used to normalize the result of all the applications of a labeled rewrite rule to a term. When evaluating a local assignment `where v:=(S) t` of an ELAN rewrite rule, the term `t` is first normalized according to the specified set of unlabeled rewrite rules and then the strategy `S` is applied to its normal form. Moreover, each time a labeled rewrite rule is applied to a term, the ELAN evaluation mechanism normalizes the result of its application with respect to the set of unlabeled rewrite rules.

Hence, the ELAN rewrite rule from Example 4.6 should be represented in the ρ-calculus by the term

$$x \to [im(\mathcal{R}_f)]([y \to [True \to next(y)] \\ ([im(\mathcal{R}_f)](nf(y)))]([\{s1 \to s2, s1 \to s3\}]([im(\mathcal{R}_f)](x))))$$

where $\mathcal{R}_f$ represents the set of (unlabeled) rewrite rules modulo which we normalize the local assignments.

4.2.2 General strategies in the local assignments

Until now we have considered in the local assignments of a rule only strategies that do not use the respective rewrite rule. The representation of an ELAN rule with local calls to strategies defined by using this rule must be parameterized by the definition of the respective strategies. For example, a rule with local assignments of the form

$$[label] \qquad l \Longrightarrow r \ \texttt{where}\ x := (s)t$$

is represented by the ρ-term

$$label(f) \triangleq l \to [x \to r]([[f](s)](t))$$

where the free variable f will be instantiated by the set of strategies of the program containing the rule labeled by $label$.

4.2.3 ELAN strategies and programs

The elementary ELAN strategies has, in most of the cases, a direct representation in the ρ-calculus. The identity (`id`) and the failure (`fail`) as well as the concatenation (`;`) are directly represented in the ρ-calculus by the ρ-operators id, $fail$ and ";" respectively, defined in Section 2.1. The strategy $\texttt{dk}(S_1, \ldots, S_n)$ is represented in the ρ-calculus by the set $\{S_1, \ldots, S_n\}$ and the strategy $\texttt{first}(S_1, \ldots, S_n)$ by the ρ-term $first(S_1, \ldots, S_n)$ defined in Section 2.2. The iteration strategy operator `repeat*` is easily represented by using the ρ-operator $repeat*$.

Strategies can be used in the evaluation of the local assignments and these strategies are expressed using rewrite rules. Therefore, the ELAN strategies can be represented by ρ-terms in the same way as the ELAN rewrite rules.

Example 4.8 The ELAN strategy `attStrat` used in Example 4.3 is immediately represented by the ρ-term

$$attStrat_\rho \to repeat{*}(\{initiate_\rho, \ldots, intruder_\rho\}); attackFound_\rho$$

where we suppose that $initiate_\rho$, $intruder_\rho$, $attackFound_\rho$ are the representations in ρ-calculus of the corresponding ELAN strategies.

For the representation of the user-defined strategies in an ELAN program we use an approach based on the fixed-point operator and similar to that used in the case of conditional rules in Section 3.2. If we consider an ELAN program containing the strategies $S_1, \ldots, S_n$ and a set of labeled rules, then the ρ-term representing the program is

$$P \triangleq [\Theta](S)$$

where

$$S \triangleq f \to (y \to [\{S_i \to Body_i \mid i = 1 \ldots n\}](y))$$

and $Body_i$ represent the right-hand sides of the strategies with each strategy S_i replaced by $[f](S_i)$, each rule label replaced by the ρ-representation of the rule and each ELAN strategy operator replaced by its correspondent in the ρ-calculus.

To sum-up, we present the transformation of an ELAN program in a ρ-term.

Definition 4.9 We consider an ELAN without importations.

1. The signature of the corresponding ρ-calculus is obtained from the operator declarations of the ELAN program.
2. Starting from unlabeled rules of the form

   $[]\quad l_i(\overline{x}) \Longrightarrow r_i(\overline{x}, \overline{y})$

   `where` $(sort)\ u_i(\overline{y}) := ()t_i(\overline{x})$

   `if` $c_i(\overline{x}, \overline{y})$

   `end`

   we build the term

   $$R_{nn} \triangleq f \to (z \to [im(\{l_i(\overline{x}) \to [u_i(\overline{y}) \to [True \to r_i(\overline{x}, \overline{y})]([f](c_i(\overline{x}, \overline{y})))](t_i(\overline{x})) \mid i = 1 \ldots n\})](z))$$

   The *innermost* normalization w.r.t. the set of unlabeled rules is represented by the term

   $$IM_{nn} \triangleq [\Theta](R_{nn})$$

   The encoding is extended in an incremental way to rules containing several conditions and local assignments. The encoding can be simplified if the program does not contain unlabeled conditional rules; in this case the term IM_{nn} becomes

   $$IM_{nn} \triangleq im(\{l_i(\overline{x}) \to [u_i(\overline{y}) \to r_i(\overline{x}, \overline{y})](t_i(\overline{x})) \mid i = 1 \ldots n\})$$

   where the rules with local assignments can be simplified to elementary rules.

3. For each labeled rule of the form

$$[label] \quad l(\overline{x}) \Longrightarrow r(\overline{x},\overline{y})$$

```
                    where (sort) u(ȳ) := (s)t(x̄)
                    if c(x̄,ȳ)
end
```

we build the term

$$label(f) \triangleq \; f \to (l(\overline{x}) \to [IM_{nn}](\; [u(\overline{y}) \to [True \to r(\overline{x},\overline{y})]([IM_{nn}](c(\overline{x},\overline{y})))]([[f](s)]([IM_{nn}](t(\overline{x}))))))$$

4. For each strategy of the form

```
[]    S ⟹ Body
end
```

we build the term

$$S \to BodyRho(f)$$

where $BodyRho$ represents the right-hand side $Body$ of the strategy with each strategy symbol S_i replaced by $[f](S_i)$, each rule label $label$ replaced by the ρ-representation $label(f)$ of the rule and each ELAN strategy operator replaced by its correspondent in the ρ-calculus.

The ELAN program defining the strategies $S_1, \ldots, S_n$ is represented by the ρ-term

$$P \triangleq [\Theta](S)$$

where

$$S \triangleq f \to (z \to [\{S_i \to BodyRho_i(f) \mid i = 1 \ldots n\}](z))$$

and $BodyRho_i(f)$ represents the encoding of the strategy S_i.

The application of a strategy $\mathcal{S}$ of an ELAN program $\mathcal{P}$ to a term t is represented by the ρ-term $[[P](s)](t)$ where P is the ρ-term representing the program $\mathcal{P}$ and s is the name of the strategy $\mathcal{S}$. If the execution of the program $\mathcal{P}$ for evaluating the term t according to the strategy $\mathcal{S}$ leads to the results $u_1, \ldots, u_n$, then the ρ-term $[[P](s)](t)$ is reduced to the set term $\{u_1, \ldots, u_n\}$.

In Example 4.10 we present an ELAN module and the ρ-interpretations of all the rules and strategies and thus, of the ELAN program.

Example 4.10 The module `automaton` describes an automaton with the states `s1,s2,s3,s4,s5` and with the non-deterministic transitions described by a set of rules containing the rules labeled with `r12,r13,r25,r32,r34,r41`. The operator `next` defines the next state in a deterministic manner and its behavior is described by a set of unlabeled rules. The states can be "final" (`final`) or "closed" (`closed`). The double transitions with an intermediate non-final and non-closed state are described by the rules `double_f` and respectively `double_c`.

```
module  automaton
import global bool;end
sort state ;end
operators global
    s1,s2,s3,s4,s5 : state;
    next(@) : (state)  state;
    final(@) : (state)  bool;
    closed(@) : (state)  bool;
end
stratop  global
    follow   : <state -> state>   bs;
    gen_double   : <state -> state>   bs;
    cond_double   : <state -> state>   bs;
end
rules for bool
global
 [] final(s_1) => false   end         [] closed(s_1) => false   end
 [] final(s_2) => true    end         [] closed(s_2) => false   end
 [] final(s_3) => false   end         [] closed(s_3) => true    end
 [] final(s_4) => false   end         [] closed(s_4) => true    end
 [] final(s_5) => true    end         [] closed(s_5) => true    end
end
rules for state
    x,y : state;
global
  [r12] s1 => s2   end                  [] next(s1) => s3   end
  [r13] s1 => s3   end                  [] next(s2) => s5   end
  [r25] s2 => s5   end                  [] next(s3) => s2   end
  [r32] s3 => s2   end                  [] next(s4) => s1   end
  [r34] s3 => s4   end                  [] next(s5) => s5   end
  [r41] s4 => s1   end

  [double_f]  x => next(y)
                      where y := (follow) x
                      if not final(y)                end
  [double_c]  x => next(y)
                      where y := (follow) x
                      if not closed(y)               end
end

strategies for state
implicit
    []follow         =>   dk(r12,r13,r25,r32,r34,r41)            end
    []gen_double     =>   follow;follow                          end
    []cond_double    =>   dk(double_f,double_c)                  end
end
end
```

We denote by B the set of unlabeled rules defined in the imported modules `bool` and describing operations on booleans.

The set of unlabeled rules from the module `automaton` are represented by the ρ-term

$$R \triangleq \{next(s1) \to s3, \ldots, next(s5) \to s5, \\ final(s1) \to false, \ldots, final(s5) \to true, \\ closed(s1) \to false, \ldots, closed(s5) \to true\}$$

and we note $RC = R \cup B$.

The rules labeled with `double_f` and `double_c` are represented by the ρ-rules

$$double_f(f) \triangleq x \to [im(RC)]([y \to [True \to next(y)]([im(RC)](not\ final(y)))] \\ ([[f](follow)]([im(RC)](x))))$$

and respectively

$$double_c(f) \triangleq x \to [im(RC)]([y \to [True \to next(y)]([im(RC)](not\ closed(y)))] \\ ([[f](follow)]([im(RC)](x))))$$

The strategies from the module `automaton` are represented by the ρ-terms

$$follow \triangleq follow \to \{s1 \to s2, s1 \to s3, s2 \to s5, s3 \to s2, s3 \to s4, s4 \to s1\}$$
$$gen_double(f) \triangleq gen_double \to [f](follow); [f](follow)$$
$$cond_double(f) \triangleq cond_double \to \{double_f(f), double_c(f)\}$$

and we obtain the term representing the ELAN program `automaton`

$$automaton \triangleq [\Theta](S)$$

where

$$S \triangleq f \to (y \to [\{follow, gen_double(f), cond_double(f)\}](y))$$

The execution of the program `automaton` for evaluating the term `s1` with the strategy `cond_double` corresponds to the reduction of the term

$$[[automaton](cond_double)](s1)$$

In ELAN, we obtain for such an execution the results `2` and `5` and the reduction of the corresponding ρ-term leads to the set $\{2, 5\}$.

In Example 4.10 we presented a relatively simple ELAN module but, representative for the main features of the ELAN language. Following the same methodology, more complicated rules and strategies can be handled.

Notice that this provides, in particular, a very precise description of all the rewriting primitives, including the semantics of the conditional rewriting used by the language. To the best of our knowledge, this is the first *explicit* and *full* description of a rewrite based programming language.

5 Conclusion

Using the ρ^{1st}-calculus, an extension of the ρ-calculus, appropriate definitions of term traversal operators and of a fixed-point operator can be given. This enables us to apply repeatedly a (set of) rewrite rule(s) and consequently to define a ρ-term representing the normalization according to a set of rewrite rules. Starting from this representation we showed how the ρ^{1st}-calculus can be used to define conditional rewriting and to give a semantics to ELAN modules. Of course, this could be applied to many other frameworks, including rewrite based languages like ASF+SDF, ML, Maude, Stratego or CafeOBJ but also production systems and non-deterministic transition systems.

Starting from these first results on the rewriting calculus, we have already explored, in subsequent papers, two different directions: the ρ-calculus with explicit substitutions and typed rewriting calculi. In [9] we have proposed a version of the calculus where the substitution application is described at the same level as the other evaluation rules. Starting from the λ-calculus with explicit substitutions, and in particular the $\lambda\sigma_{\Uparrow}$-calculus, we developed the ρ-calculus with explicit substitutions, called the $\rho\sigma$-calculus and we showed that the $\rho\sigma$-calculus is confluent under the same conditions as the $\rho_{\emptyset}$-calculus. Indeed, what makes the explicit substitution setting even more interesting than in the case of λ-calculus is that not only the substitution and therefore renaming mechanism is handled explicitly, but the substitution itself is explicitly represented. This is extremely useful since computing a substitution could be very expensive like for associativity-commutativity where the matching algorithm is exponential in the size of terms. Moreover, since a derivation may fail (like when searching for the right instance of a conditional), memorizing the substitution is mandatory. This allows us in particular to use the ρ-calculus with explicit substitutions as the language to describe proof terms of ELAN computations.

The ρ-calculus is not terminating in the untyped case. In order to recover this property we have imposed in [11] a more strict discipline on the ρ-term formation by introducing a type for each term. We presented a type system for the $\rho_{\emptyset}$-calculus and we showed that it has the subject reduction and strong normalization properties, *i.e.* that the reduction of any well-typed term is terminating and preserves the type of the initial term. Additionally, we have given a new presentation *à la Church* to the ρ-calculus [13], together with nine (8+1) type systems which can be placed in a *ρ-cube* that extends the λ-cube of Barendregt. Quite interestingly, this typed calculus uses only one abstractor, namely the rule arrow. It provides therefore a solution to the identification of the λ and Π abstractors.

We used the sets to represent the non-determinism and we mentioned that other structures can be used. For example, if we want to represent all the results of an application and not only the different results, then multisets must be used and if the order of the results is significant, then a list structure is more suitable. We have thus started the study of another description of the ρ-calculus having as parameter not only the matching theory but also the structure used for the results and we have already shown its expressive power [12]. More precisely, we analyzed the correspondence between the ρ-calculus and two object oriented calculi: the *"Object Calculus"* of Abadi and Cardelli [1] and the *"Lambda Calculus of Objects"* of Fisher, Honsell and Mitchell [16]. The approach that we proposed allows the representation of objects in the style of the two mentioned calculi but also of more elaborate objects whose

behavior is described by using the matching power.

As a new emergent framework, the ρ_T-calculus offers an original view point on rewriting and higher-order logic and it opens new challenges to further understand related topics. First, to go further in the study and the use of the ρ_T-calculus for the combination of first-order and higher-order paradigms, the investigation of the relationship of this calculus with higher-order rewrite concepts like CRS and HOR [30] should be deepened. Second, several directions should be investigated, amongst them, we can mention the following:

- The analysis of the properties of the ρ_T-calculus with a matching theory T more elaborate than syntactic matching.
- A generic description of the conditions which must be imposed for the matching theory T in order to obtain the confluence and the termination of the ρ_T-calculus should be defined and then, show that these conditions are satisfied for particular theories such as associativity and commutativity.
- The models of the rewriting calculus should be defined, studied and compared with the ones of the algebraic as well as higher-order structures.
- As mentioned previously, we conjecture that the ρ^{1st}-calculus can not be expressed in the ρ-calculus because of the semantics of the empty set as rule application failure.

Finally, from the practical point of view, the various instances of the ρ-calculus must be further implemented and used as rewriting tools. We have already realized an implementation in ELAN of the $\rho_\emptyset$-calculus and we experimented with various evaluation strategies. This implementation could be further used in order to define object oriented paradigms. Dually, an object oriented version of the ELAN language has been realized [14], with a semantics given by the rewriting calculus.

This shows that this new calculus is very attractive in terms of semantics as well as unifying capabilities and we believe that it can serve as a basic tool for the integration of semantic and logical frameworks.

Acknowledgments

We would like to thank Hélène Kirchner, Pierre-Etienne Moreau and Christophe Ringeissen from the Protheo Team for the useful interactions we had on the topics of this paper, Vincent van Oostrom for suggestions and pointers to the literature, Roberto Bruni and David Wolfram for their detailed and very useful comments on a preliminary version of this work and Delia Kesner for fruitful discussions. We are grateful to Luigi Liquori for many comments and exciting discussions on the ρ-calculus and its applications. Many thanks also to Thérèse Hardin and Nachum Dershowitz for their interest, encouragements and helpful suggestions for improvement. Finally special thanks are due to the referees for the very complete and careful reading of the paper as well as constructive and useful remarks.

References

[1] M. Abadi and L. Cardelli. *A Theory of Objects.* Springer Verlag, 1996.

[2] H. P. Barendregt. *The Lambda-Calculus, its syntax and semantics.* Studies in Logic and the Foundation of Mathematics. Elsevier Science Publishers B. V. (North-Holland), Amsterdam, 1984. Second edition.

[3] P. Borovanský, C. Kirchner, H. Kirchner, P.-E. Moreau, and M. Vittek. ELAN: A logical framework based on computational systems. In J. Meseguer, editor, *Proceedings of the first international workshop on rewriting logic*, volume 4 of *Electronic Notes in TCS*, Asilomar (California), September 1996.

[4] P. Borovanský, C. Kirchner, H. Kirchner, and P.-E. Moreau. *ELAN* from the rewriting logic point of view. Research report, LORIA, November 1999.

[5] A. Bouhoula and M. Rusinowitch. Implicit induction in conditional theories. *Journal of Automated Reasoning*, 14(2):189–235, 1995.

[6] C. Castro. *Une approche déductive de la résolution de problèmes de satisfaction de contraintes.* Thèse de Doctorat d'Université, Université Henri Poincaré – Nancy 1, France, 1998.

[7] M. Clavel, S. Eker, P. Lincoln, and J. Meseguer. Principles of Maude. In J. Meseguer, editor, *Proceedings of the first international workshop on rewriting logic*, volume 4, Asilomar (California), September 1996. Electronic Notes in Theoretical Computer Science.

[8] H. Cirstea. Specifying Authentication Protocols Using ELAN. In *Workshop on Modelling and Verification*, Besancon, France, December 1999.

[9] H. Cirstea. *Calcul de réécriture : fondements et applications.* Thèse de Doctorat d'Université, Université Henri Poincaré - Nancy I, 2000.

[10] H. Cirstea and C. Kirchner. Theorem Proving Using Computational Systems: The Case of the B Predicate Prover. In *Workshop CCL'97*, Schloβ Dagstuhl, Germany, September 1997.

[11] H. Cirstea and C. Kirchner. The Simply Typed Rewriting Calculus. In *3rd International Workshop on Rewriting Logic and its Applications*, Kanazawa, Japan, September 2000. Electronic Notes in Theoretical Computer Science.

[12] H. Cirstea, C. Kirchner, and L. Liquori. Matching Power. In A. Middeldorp, editor, *Proceedings of RTA'2001*, Lecture Notes in Computer Science, Utrecht (The Netherlands), May 2001. Springer-Verlag.

[13] H. Cirstea, C. Kirchner, and L. Liquori. The Rho Cube. In F. Honsell, editor, *Foundations of Software Science and Computation Structures*, Lecture Notes in Computer Science, Genova, Italy, April 2001.

[14] H. Dubois and H. Kirchner. Objects, rules and strategies in ELAN. In *Proceedings of the second AMAST workshop on Algebraic Methods in Language Processing, Iowa City, Iowa, USA*, May 2000.

[15] N. Dershowitz and M. Okada. A rationale for conditional equational programming. *Theoretical Computer Science*, 75:111–138, 1990.

[16] K. Fisher, F. Honsell, and J. C. Mitchell. A Lambda Calculus of Objects and Method Specializatio n. *Nordic Journal of Computing*, 1(1):3–37, 1994.

[17] K. Futatsugi and A. Nakagawa. An overview of CAFE specification environment – an algebraic approach for creating, verifying, and maintaining formal specifications over networks. In *Proceedings of the 1st IEEE Int. Conference on Formal Engineering Methods*, 1997.

[18] M. Gordon, A. Milner, and C. Wadsworth. *Edinburgh LCF: A Mechanized Logic of Computation*, volume 78 of *Lecture Notes in Computer Science*. Springer-Verlag, New York (NY, USA), 1979.

[19] C. Kirchner, H. Kirchner, and M. Vittek. Designing constraint logic programming languages using computational systems. In P. Van Hentenryck and V. Saraswat, editors, *Principles and Practice of Constraint Programming. The Newport Papers.*, chapter 8, pages 131–158. The MIT press, 1995.

[20] P. Klint. The ASF+SDF Meta-environment User's Guide. Technical report, CWI, 1993.

[21] C. Kirchner and C. Ringeissen. Rule-Based Constraint Programming. *Fundamenta Informaticae*, 34(3):225–262, September 1998.

[22] J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.

[23] R. Milner. A proposal for standard ML. In *Proceedings ACM Conference on LISP and Functional Programming*, 1984.

[24] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[25] Protheo Team. The ELAN home page. WWW Page, 2001. `http://elan.loria.fr`.

[26] A. M. Turing. The $\wp$-functions in λ-K-conversion. *The Journal of Symbolic Logic*, 2:164, 1937.

[27] E. Visser and Z. el Abidine Benaissa. A core language for rewriting. In C. Kirchner and H. Kirchner, editors, *Proceedings of the second International Workshop on Rewriting Logic and Applications*, volume 15, `http://www.elsevier.nl/locate/entcs/volume16.html`, Pont-à-Mousson (France), September 1998. Electronic Notes in Theoretical Computer Science.

[28] E. Visser. Strategic pattern matching. In P. Narendran and M. Rusinowitch, editors, *Rewriting Techniques and Applications (RTA'99)*, volume 1631 of *Lecture Notes in Computer Science*, pages 30–44, Trento, Italy, July 1999. Springer-Verlag.

[29] M. Vittek. ELAN*: Un cadre logique pour le prototypage de langages de programmation avec contraintes.* Thèse de Doctorat d'Université, Université Henri Poincaré – Nancy 1, October 1994.

[30] V. van Oostrom and F. van Raamsdonk. Comparing combinatory reduction systems and higher-order rewrite systems. In *HOA'93*, volume 816 of *Lecture Notes in Computer Science*, pages 276–304. Springer-Verlag, 1993.

Received October 1, 2000. Revised: January 26, 2001, February 9, 2001

Tableau Reasoning and Programming with Dynamic First Order Logic

Jan van Eijck, *CWI and ILLC, Amsterdam, E-mail: jve@cwi.nl.*

Juan Heguiabehere, *ILLC, Amsterdam, E-mail: juanh@wins.uva.nl.*

Breanndán Ó Nualláin, *ILLC, Amsterdam, E-mail: bon@illc.uva.nl.*

Abstract

Dynamic First Order Logic (DFOL) results from interpreting quantification over a variable v as change of valuation over the v position, conjunction as sequential composition, disjunction as non-deterministic choice, and negation as (negated) test for continuation. We present a tableau style calculus for DFOL with explicit (simultaneous) binding, prove its soundness and completeness, and point out its relevance for programming with DFOL, for automated program analysis including loop invariant detection, and for semantics of natural language. We also extend this to an infinitary calculus for DFOL with iteration and connect up with other work in dynamic logic.

Keywords: Dynamic Logic, First Order Logic, Assertion Calculus, Tableau Reasoning

1 Introduction

The language we use and analyze in this paper consists of formulas that can be used both for programming and for making assertions about programs. The only difference between a program and an assertion is that an assertion is a program with its further computational effect blocked off. In the notation we will introduce below: if ϕ is a program, then $(\!(\phi)\!)$ is the assertion that the program ϕ can be executed. Execution of ϕ will in general lead to a set of computed answer bindings, execution of $(\!(\phi)\!)$ to a yes/no answer indicating success or failure of ϕ.

Since the formulas of our language, DFOL, can be used for description and computation alike, our calculus is both an execution mechanism for DFOL and a tool for theorem proving in DFOL. One of the benefits of mixing calculation and assertion is that the calculus can be put to use to automatically derive assertions about programs for purposes of verification. And since DFOL has its roots in Natural Language processing (just as Prolog does), we also see a future for our tool-set in a computational semantics of natural language.

We start our enterprise by developing a theory of binding for DFOL that we then put to use in a calculus for DFOL with explicit binding. The explicit bindings represent the intermediate results of calculation that get carried along in the computation process. We illustrate with examples from standard first order reasoning, natural language processing, imperative programming, and derivation of postconditions for imperative programs.

 ©Oxford University Press

Finally, we develop an infinitary calculus for DFOL plus iteration, with a completeness proof. Details of the relationships with existing calculi are given below. The two calculi that are the subject of this paper form the computation and inference engine of a toy programming language for theorem proving and computing with DFOL, *Dynamo*.

2 Dynamic First Order Logic

Dynamic First Order Logic results from interpreting quantification over v as change of valuation over the v position, conjunction as sequential composition, disjunction as nondeterministic choice, and negation as (negated) test for continuation. See Groenendijk and Stokhof [16] for a presentation and Visser [31] for an in-depth analysis. A sound and complete sequent style calculus for DFOL (without choice) was presented in Van Eijck [12]. In this paper we present a calculus that also covers the choice operator, and that is much closer to standard analytic tableau style reasoning for FOL (see Smullyan [29] for a classical presentation, Fitting [13] for a textbook treatment and connections with automated theorem proving, [17] for an excellent overview, and [8] for an encyclopedic account).

For applications of DFOL to programming, the presence of the choice operation $\cup$ in the language is crucial: choice is the basis of 'if then else', and of all nondeterministic programming constructs for exploring various avenues towards a solution. It can (and has been) argued that the full expressive power of $\cup$ is not necessary for applications of DFOL to natural language semantics. In fact, the presentation of dynamic predicate logic (DPL) in [16] does not cover $\cup$: in DPL, choice is handled in terms of negation and conjunction, with the argument that natural language 'or' is externally static. This means that an 'or' construction behaves like a test. The present calculus deals with DFOL including choice.

A very convenient extension that we immediately add to DFOL is representation of simultaneous binding. It is well known that bindings or substitutions are definable in DFOL. Still we will consider them as operators in their own right, in the spirit of Venema [30], where substitutions are studied as modal operators. Simultaneous bindings can in general not be expressed in terms of single bindings without introducing auxiliary variables. E.g., the swap of variables x and y in the simultaneous binding $[y/x, x/y]$ can only be expressed as a sequence of single bindings at the expense of availing ourselves of an extra variable z, as $z := x; x := y; y := z$. The dynamic effect of this sequence of single bindings is not quite the same as that of $[y/x, x/y]$, for $z := x; x := y; y := z$ changes the value of z, while $[y/x, x/y]$ does not, and the semantics of DFOL is sensitive to such subtle differences.

A first order signature Σ is a pair $\langle P_\Sigma, F_\Sigma \rangle$, with P_Σ a set of predicate constants and F_Σ a set of function constants. Let V be an infinite set of variables, and let $a : (P_\Sigma \cup F_\Sigma) \to \mathbb{N}$ be a function that assigns to every predicate or function symbol its arity. The function symbols with arity 0 are the individual constants. The set T_Σ of terms over the signature is given in the familiar way, by $t ::= v \mid ft_1 \cdots t_n$, where v ranges over V and f over F_Σ, with $a(f) = n$. The sub-terms of a term are given as usual. We will write sequences of terms $t_1, \ldots, t_n$ as $\bar{t}$.

A binding θ is a function $V \to T_\Sigma$ that makes only a finite number of changes, i.e., θ has the property that $dom(\theta) = \{v \in V \mid \theta(v) \neq v\}$ is finite. See Apt [1] and

Doets [10] for lucid introductions to the subject of binding in the context of logic programming. We will use $rng(\theta)$ for $\{\theta(v) \in T_\Sigma \mid \theta(v) \neq v\}$, and $var(rng(\theta))$ for $\cup\{var(\theta(v)) \mid v \in dom(\theta)\}$, where $var(t)$ is the set of variables occurring as a subterm in t. An explicit form (or: a representation) for binding θ is a sequence

$$[\theta(v_1)/v_1, \ldots, \theta(v_n)/v_n],$$

where $\{v_1, \ldots, v_n\} = dom(\theta)$, (i.e., $\theta(v_i) \neq v_i$, for only the *changes* are listed), and $i \neq j$ implies $v_i \neq v_j$ (i.e., each variable in the domain is mentioned only once). We will use [] for the binding that changes nothing, i.e, [] is the only binding θ with $dom(\theta) = \emptyset$. We use θ, ρ, possibly with indices, as meta-variables ranging over bindings. Representations for bindings are given, as usual, by:

$$\theta \quad ::= \quad [] \mid [t_1/v_1, \ldots, t_n/v_n] \qquad \text{provided } t_i \neq v_i, \text{ and } v_i = v_j \text{ implies } i = j.$$

We let $\circ$ denote the syntactic operation of composition of binding representations:

> Let $\theta = [t_1/v_1, \ldots, t_n/v_n]$ and $\rho = [r_1/w_1, \ldots, r_m/w_m]$ be binding representations. Then $\theta \circ \rho$ is the result of removing from the sequence
>
> $$[\theta(r_1)/w_1, \ldots, \theta(r_m)/w_m, t_1/v_1, \ldots, t_n/v_n]$$
>
> the binding pairs $\theta(r_i)/w_i$ for which $\theta(r_i) = w_i$, and the binding pairs t_j/v_j for which $v_j \in \{w_1, \ldots, w_m\}$.

For example, $[x/y] \circ [y/z] = [x/z, x/y]$, $[x/z, y/x] \circ [z/x] = [x/z]$.

We are now in a position to define the DFOL language $\mathcal{L}_\Sigma$ over signature Σ. We distinguish between DFOL units and DFOL formulas (or sequences).

Definition 2.1 (The DFOL language $\mathcal{L}_\Sigma$ over signature Σ)

$$\begin{array}{rcl} t & ::= & v \mid f\bar{t} \\ U & ::= & \theta \mid \exists v \mid P\bar{t} \mid t_1 \doteq t_2 \mid \neg(\phi) \mid (\phi_1 \cup \phi_2) \end{array}$$

We will omit parentheses where it doesn't create syntactic ambiguity, and allow the usual abbreviations: we write $\perp$ for $\neg([])$, $\neg P\bar{t}$ for $\neg(P\bar{t})$, $t_1 \neq t_2$ for $\neg(t_1 \doteq t_2)$, $\phi_1 \cup \phi_2$ for $(\phi_1 \cup \phi_2)$. Similarly, $(\phi \rightarrow \psi)$ abbreviates $\neg(\phi; \neg(\psi))$, $\forall v(\phi)$ abbreviates $\neg(\exists v; \neg(\phi))$. A formula ϕ is a literal if ϕ is of the form $P\bar{t}$ or $\neg P\bar{t}$, or of the form $t_1 \doteq t_2$ or $t_1 \neq t_2$. The complement $\overline{\phi}$ of a formula ϕ is given by: $\overline{\phi} := \psi$ if ϕ has the form $\neg(\psi)$ and $\overline{\phi} := \neg(\phi)$ otherwise. We abbreviate $\neg\neg(\phi)$ as $(\!(\phi)\!)$, and we will call formulas of the form $(\!(\phi)\!)$ *block* formulas.

We can think of formula ϕ as built up from units U by concatenation. For formula induction arguments, it is sometimes convenient to read a unit U as the formula $U; []$ (recall that [] is the empty binding), thus using [] for the empty list formula. In other words, we will silently add the [] at the end of a formula list when we need its presence in recursive definitions or induction arguments on formula structure.

Given a first order model $\mathcal{M} = (D, I)$ for signature Σ, the semantics of DFOL language $\mathcal{L}_\Sigma$ is given as a binary relation on the set ${}^V D$, the set of all variable maps (variable states, valuations) into the domain of the model. We impose the usual non-empty domain constraint of FOL: any Σ model $\mathcal{M} = (D, I)$ has $D \neq \emptyset$. If $s, u \in {}^V D$,

we use $s \sim_v u$ to indicate that s, u differ at most in their value for v, and $s \sim_X u$ to indicate that s, u differ at most in their values for the members of X. If $s \in {}^V D$ and $v, v' \in V$, we use $s[v'/v]$ for the valuation u given by $u(v) = s(v')$, and $u(w) = s(w)$ for all $w \in V$ with $w \neq v$. Also, if s and v are as before and $d \in D$ we use $s[d/v]$ for the valuation u given by $u(v) = d$, and $u(w) = s(w)$ for all $w \in V$ with $w \neq v$.

$\mathcal{M} \models_s P\bar{t}$ indicates that s satisfies the predicate $P\bar{t}$ in $\mathcal{M}$ according to the standard truth definition for classical first order logic. $[\![t]\!]_s^{\mathcal{M}}$ gives the denotation of t in $\mathcal{M}$ under s. If θ is a binding and s a valuation (a member of ${}^V D$), we will use s_θ for the valuation u given by $u(v) = [\![\theta(v)]\!]_s^M$.

Definition 2.2 (Semantics of DFOL)

$$
\begin{array}{rll}
{}_s[\![\theta]\!]_u^{\mathcal{M}} & \textit{iff} & u = s_\theta \\
{}_s[\![\exists v]\!]_u^{\mathcal{M}} & \textit{iff} & s \sim_v u \\
{}_s[\![P\bar{t}]\!]_u^{\mathcal{M}} & \textit{iff} & s = u \textit{ and } \mathcal{M} \models_s P\bar{t} \\
{}_s[\![t_1 \doteq t_2]\!]_u^{\mathcal{M}} & \textit{iff} & s = u \textit{ and } [\![t_1]\!]_s^{\mathcal{M}} = [\![t_2]\!]_s^{\mathcal{M}} \\
{}_s[\![\neg(\phi)]\!]_u^{\mathcal{M}} & \textit{iff} & s = u \textit{ and there is no } u' \textit{ with } {}_s[\![\phi]\!]_{u'}^{\mathcal{M}} \\
{}_s[\![\phi_1 \cup \phi_2]\!]_u^{\mathcal{M}} & \textit{iff} & {}_s[\![\phi_1]\!]_u^{\mathcal{M}} \textit{ or } {}_s[\![\phi_2]\!]_u^{\mathcal{M}} \\
{}_s[\![U;\phi]\!]_u^{\mathcal{M}} & \textit{iff} & \textit{there is a } u' \textit{ with } {}_s[\![U]\!]_{u'}^{\mathcal{M}} \textit{ and } {}_{u'}[\![\phi]\!]_u^{\mathcal{M}}
\end{array}
$$

Note that it follows from this definition that

$$
{}_s[\![(\!(\phi)\!)]\!]_u^{\mathcal{M}} \text{ iff } s = u \text{ and there is a } u' \text{ with } {}_s[\![\phi]\!]_{u'}^{\mathcal{M}}.
$$

Thus, block formulas have their dynamic effects blocked off: double negation transforms the semantic transition relation into a test.

We introduce a syntactic blocking operation on formulas as follows ($=$ is used for syntactic identity):

Definition 2.3 (Blocking Operation on Formulas)

$$
\begin{array}{rcl}
(\theta)^{\boxdot} & := & (\!(\theta)\!) \\
(\exists v)^{\boxdot} & := & (\!(\exists v)\!) \\
(P\bar{t})^{\boxdot} & := & P\bar{t} \\
(t_1 \doteq t_2)^{\boxdot} & := & t_1 \doteq t_2 \\
(\neg(\phi))^{\boxdot} & := & \neg(\phi) \\
(\phi_1 \cup \phi_2)^{\boxdot} & := & \begin{cases} (\phi_1 \cup \phi_2) & \textit{if } {\phi_1}^{\boxdot} = \phi_1, {\phi_2}^{\boxdot} = \phi_2, \\ (\!(\phi_1 \cup \phi_2)\!) & \textit{otherwise} \end{cases} \\
(U;\phi)^{\boxdot} & := & \begin{cases} U;\phi & \textit{if } U^{\boxdot} = U, \phi^{\boxdot} = \phi, \\ (\!(U;\phi)\!) & \textit{otherwise.} \end{cases}
\end{array}
$$

E.g., $(\exists x; Px)^{\boxdot} = (\!(\exists x; Px)\!)$, and $(\neg(\exists x; Px))^{\boxdot} = \neg(\exists x; Px)$. By induction on formula structure we get from Definitions 2.2 and 2.3 that the blocking operation makes a formula into a test, in the following sense:

Proposition 2.4 *For all $\mathcal{M}$ and all valuations s, u for $\mathcal{M}$, all $\mathcal{L}_\Sigma$ formulas ϕ: ${}_s[\![\phi^{\boxdot}]\!]_u^{\mathcal{M}}$ iff $s = u$ and there is a u' with ${}_s[\![\phi]\!]_{u'}^{\mathcal{M}}$.*

The key relation we want to get to grips with in this paper is the dynamic entailment relation that is due to [16]:

Definition 2.5 (Entailment in DFOL) *ϕ dynamically entails ψ, notation $\phi \models \psi$, :⇔ for all $\mathcal{L}_\Sigma$ models $\mathcal{M}$, all valuations s, u for $\mathcal{M}$, if ${}_s[\![\phi]\!]_u^{\mathcal{M}}$ then there is a variable state u' for which ${}_u[\![\psi]\!]_{u'}^{\mathcal{M}}$.*

3 Binding in DFOL

Bindings θ are lifted to (sequences of) terms and (sets of) formulas in the familiar way:

Definition 3.1 (Binding in DFOL)

$$\begin{array}{rcl}
\theta(ft_1 \cdots t_n) & := & f\theta(t_1)\cdots\theta(t_n) \\
\theta(t_1, \ldots, t_n) & := & \theta(t_1), \ldots, \theta(t_n) \\
\theta(\rho) & := & \theta \circ \rho \\
\theta(\rho; \phi) & := & (\theta \circ \rho)\phi \\
\theta(\exists v; \phi) & := & \exists v; \theta'\phi \text{ where } \theta' = \theta \backslash \{t/v \mid t \in T\} \\
\theta(P\bar{t}; \phi) & := & P\theta\bar{t}; \theta\phi \\
\theta(t_1 \doteq t_2; \phi) & := & \theta t_1 \doteq \theta t_2; \theta\phi \\
\theta((\phi_1 \cup \phi_2); \phi_3) & := & \theta(\phi_1; \phi_3) \cup \theta(\phi_2; \phi_3) \\
\theta(\neg(\phi_1); \phi_2) & := & \neg(\theta\phi_1); \theta\phi_2 \\
\theta(\{\phi_1, \ldots, \phi_n\}) & := & \{\theta(\phi_1), \ldots, \theta(\phi_n)\}
\end{array}$$

Note that it follows from this definition that

$$\theta((\!(\phi_1)\!); \phi_2) = (\!(\theta\phi_1)\!); \theta\phi_2.$$

Thus, binding distributes over block: this accounts for how $(\!(\cdots)\!)$ insulates dynamic binding effects.[1]

The composition $\theta \cdot \rho$ of two bindings θ and ρ has its usual meaning of 'θ after ρ', which we get by means of $\theta \cdot \rho(v) := \theta(\rho(v))$. It can be proved in the usual way, by induction on term structure, that the definition has the desired effect, in the sense that for all $t \in T$, for all binding representations θ, ρ: $(\theta \circ \rho)(t) = \theta(\rho(t)) = (\theta \cdot \rho)(t)$.

Here is an example of how to apply a binding to a formula:

$$\begin{array}{rcl}
& [a/x]Px; (Qx \cup \exists x; \neg Px); Sx \;=\; Pa; [a/x](Qx \cup \exists x; \neg Px); Sx & \\
= & Pa; ([a/x]Qx; Sx \cup [a/x]\exists x; \neg Px; Sx) \;=\; Pa; (Qa; Sa; [a/x] \cup \exists x; \neg Px; Sx) &
\end{array}$$

The binding definition for DFOL fleshes out what has been called the 'folklore idea in dynamic logic' (Van Benthem [6]) that syntactic binding $[t/v]$ works semantically as

[1] Our reasons, by the way, for preferring prefix notation for application of bindings over the more usual postfix notation have to do with the fact that in the rules of our calculus bindings have an effect on formulas on their right.

the program instruction $v := t$ (Goldblatt [15]), with semantics given by ${}_s[\![v := t]\!]_u^{\mathcal{M}}$ iff $u = s[[\![t]\!]_s^M/v]$. To see the connection, note that $v := t$ can be viewed as DFOL shorthand for $\exists v; v = t$, on the assumption that $v \notin var(t)$.

In standard first order logic, sometimes it is not safe to apply a binding to a formula, because it leads to accidental capture of free variables. The same applies here. Applying binding $[x/y]$ to $\exists x; Rxy$ is not safe, as it would lead to accidental capture of the free variable y. The following definition defines safety of binding.

Definition 3.2 (Binding θ is safe for ϕ)

$$
\begin{array}{rcl}
\theta \textit{ is safe for } \rho & & \textit{always} \\
\theta \textit{ is safe for } \rho; \phi & :\Longleftrightarrow & \theta \circ \rho \textit{ is safe for } \phi \\
\theta \textit{ is safe for } P\bar{t}; \phi & :\Longleftrightarrow & \theta \textit{ is safe for } \phi \\
\theta \textit{ is safe for } t_1 \doteq t_2; \phi & :\Longleftrightarrow & \theta \textit{ is safe for } \phi \\
\theta \textit{ is safe for } \exists v; \phi & :\Longleftrightarrow & v \notin var(rng\, \theta') \textit{ and } \theta' \textit{ is safe for } \phi \\
 & & \quad \textit{where } \theta' = \theta \backslash \{(v,t) \mid t \in T\} \\
\theta \textit{ is safe for } \neg(\phi_1); \phi_2 & :\Longleftrightarrow & \theta \textit{ is safe for } \phi_1 \textit{ and } \theta \textit{ is safe for } \phi_2 \\
\theta \textit{ is safe for } (\phi_1 \cup \phi_2); \phi_3 & :\Longleftrightarrow & \theta \textit{ is safe for } \phi_1; \phi_3 \textit{ and } \theta \textit{ is safe for } \phi_2; \phi_3
\end{array}
$$

Note that there are ϕ with [] not safe for ϕ. E.g., [] is not safe for $[y/x]\exists y; Rxy$, because $[y/x]$ is not safe for $\exists y; Rxy$. The connection between syntactic binding and semantic assignment is formally spelled out in the following:

Lemma 3.3 (Binding Lemma for DFOL) *For all Σ models $\mathcal{M}$, all $\mathcal{M}$-valuations s, u, all $\mathcal{L}_\Sigma$ formulas ϕ, all bindings θ that are safe for ϕ:*

$$ {}_s[\![\theta\phi]\!]_u^{\mathcal{M}} \textit{ iff } {}_s[\![\theta; \phi]\!]_u^{\mathcal{M}}. $$

PROOF. Induction on the structure of ϕ. ■

Immediately from this we get the following:

Proposition 3.4 *DFOL has greater expressive power than DFOL with quantification replaced by definite assignment $v := d$.*

PROOF. If ϕ is an $\mathcal{L}_\Sigma$ formula without quantifiers, every binding θ is safe for ϕ. By the binding lemma for DFOL, ϕ is equivalent to an $\mathcal{L}_\Sigma$ formula without quantifiers but with trailing bindings. It is not difficult to see that both satisfiability and validity of quantifier free $\mathcal{L}_\Sigma$ formulas with binding trails is decidable. ■

In fact, the tableau system below constitutes a decision algorithm for satisfiability or validity of quantifier free $\mathcal{L}_\Sigma$ formulas, while the trailing bindings summarize the finite changes made to input valuations.

A comparison of our definition of binding for DFOL with that of Visser [31] and [32] reveals that Visser's notion of binding follows a different intuition, namely that binding in the empty formula yields the empty formula. We think our notion is more truly dynamic, as is witnessed by the fact that it allows us to prove a binding lemma in the presence of $\cup$, which Visser's notion does not.

In the calculus we will need *input*(ϕ), the set of variables that have an input constraining occurrence in ϕ (with $\phi \in \mathcal{L}_\Sigma$), Let $var(\bar{t})$ be the variables occurring in $\bar{t}$.

Definition 3.5 (Input constrained variables of $\mathcal{L}_\Sigma$ formulas)

$$\begin{array}{rcl}
input(\theta) & := & var(rng(\theta)) \\
input(\theta;\phi) & := & var(rng(\theta)) \cup (input(\phi) \setminus dom(\theta)) \\
input(\exists v;\phi) & := & input(\phi) \setminus \{v\} \\
input(P\bar{t};\phi) & := & var(\bar{t}) \cup input(\phi) \\
input(t_1 \doteq t_2;\phi) & := & var\{t_1, t_2\} \cup input(\phi) \\
input(\neg(\phi_1);\phi_2) & := & input(\phi_1) \cup input(\phi_2) \\
input((\phi_1 \cup \phi_2);\phi_3) & := & input(\phi_1;\phi_3) \cup input(\phi_2;\phi_3).
\end{array}$$

The following proposition (the DFOL counterpart to the finiteness lemma from classical FOL) can be proved by induction on formula structure:

Proposition 3.6 *For all $\mathcal{L}_\Sigma$ models $\mathcal{M}$, all valuations s, s', u, u' for $\mathcal{M}$, all $\mathcal{L}_\Sigma$ formulas ϕ:*

$$_s[\![\phi]\!]^{\mathcal{M}}_u \text{ and } s \sim_{V \setminus input(\phi)} s' \text{ imply } \exists u' \text{ with } {}_{s'}[\![\phi]\!]^{\mathcal{M}}_{u'}.$$

4 Adaptation of Tableau Reasoning to a Dynamic Setting

We will use one-sided tableaux, with the rule for every operator o matched by a $\neg o$ rule.

In the dynamic version of FOL, order matters: the sequencing operator ';' is not commutative in general. Suppose Φ were to consist of the two formulas $\exists x; Px$ and $\neg Px$. Then if we read Φ as $\exists x; Px; \neg Px$, we get a contradiction, but if we read Φ as $\neg Px; \exists x; Px$ then the formula set has a model that contains both Ps and non-Ps.

Local Bindings Versus Global Substitutions

We will only perform a binding θ on ϕ when needed; rather than compute $\theta\phi$, the tableau rules will store $\theta; \phi$, and compute the binding in single steps as the need arises. Tableau theorem proving can be viewed as a process of gradually building a domain D and working out requirements to be imposed on that domain. The tableau procedure that investigates whether ϕ dynamically implies ψ will build a domain with positive and negative facts. For this we employ an infinite set $F_{\mathbf{sko}}$ of skolem functions, with $F_{\mathbf{sko}} \cap F_\Sigma = \emptyset$, plus a set of fresh variables $\boldsymbol{X}$, with $V \cap \boldsymbol{X} = \emptyset$. Call the extended signature Σ^*, and the extended language $\mathcal{L}_{\Sigma^*}$. Let T_{Σ^*} be the terms of the extended language, and $T^V_{\Sigma^*}$ the terms of the extended language without occurrences of members of $\boldsymbol{X}$. Call these the *frozen* terms of $\mathcal{L}_{\Sigma^*}$, and bear in mind that frozen terms, unlike ground terms, may contain occurrences of variables in V. Call an $\mathcal{L}_{\Sigma^*}$ literal *frozen* if it contains only frozen terms.

The variables in $\boldsymbol{X}$ will function as universal tableau variables [13]. While the bindings of the variables from V are local to a tableau branch, the bindings of the variables from $\boldsymbol{X}$ are global to the whole tableau. Next to the (local) bindings for the variables V of $\mathcal{L}_\Sigma$, we introduce (global) substitutions $\boldsymbol{\sigma}$ for the fresh variables $\boldsymbol{X}$ in $\mathcal{L}_{\Sigma^*}$, and extend these to (sequences of) terms and (sets of) formulas in the manner of Definition 3.1. A substitution $\boldsymbol{\sigma}$ is a unifier of a set of (sequences of) terms T if $\boldsymbol{\sigma}T$ contains a single term (sequence of terms). It is a most general unifier (MGU) of T if $\boldsymbol{\sigma}$ is a unifier of T, and for all unifiers $\boldsymbol{\rho}$ of T there is a $\boldsymbol{\theta}$ with $\boldsymbol{\sigma} = \boldsymbol{\theta} \cdot \boldsymbol{\rho}$. Similarly

for formulas. Note that only unifiers for global *substitutions* (the term maps for the global tableau variables from $\boldsymbol{X}$) will ever be computed.

The definitions and results on binding extend to bindings with values in T_{Σ^*}, and to substitutions (domain $\subset \boldsymbol{X}$, values in T_{Σ^*}). Still, the global substitutions play an altogether different rôle in the tableau construction process, so we use a different notation for them, and write (representations for) global substitutions as

$$\{\boldsymbol{x}_1 \mapsto t_1, \ldots, \boldsymbol{x}_n \mapsto t_n\}.$$

5 Tableaux for DFOL Formula Sets

If Σ is a first order signature, a DFOL tableau over Σ is a finitely branching tree with nodes consisting of (sets of) $\mathcal{L}_{\Sigma^*}$ formulas. A branch in a tableau $\boldsymbol{T}$ is a maximal path in $\boldsymbol{T}$. We will follow custom in occasionally identifying a branch B with the set of its formulas.

Let Φ be a set of $\mathcal{L}_\Sigma$ formulas. A DFOL tableau for Φ is constructed by a (possibly infinite) sequence of applications of the following rules:

Initialization The tree consisting of a single node [] is a tableau for Φ.

Binding Composition Suppose $\boldsymbol{T}$ is a tableau for Φ and B a branch in $\boldsymbol{T}$. Let $\phi \in B \cup \Phi$, let $\theta;\rho$ occur in ϕ, and let ϕ' be the result of replacing $\theta;\rho$ in ϕ by $\theta \circ \rho$. Then the tree $\boldsymbol{T}'$ constructed from $\boldsymbol{T}$ by extending B by ϕ' is a tableau for Φ.

Expansion Suppose $\boldsymbol{T}$ is a tableau for Φ and B a branch in $\boldsymbol{T}$. Let $\phi \in B \cup \Phi$. Then the tree $\boldsymbol{T}'$ constructed from $\boldsymbol{T}$ by extending B according to one of the tableau expansion rules, applied to ϕ, is a tableau for Φ.

Equality Replacement Suppose $\boldsymbol{T}$ is a tableau for Φ and B a branch in $\boldsymbol{T}$. Let $t_1 \doteq t_2 \in B \cup \Phi$ or $t_2 \doteq t_1 \in B \cup \Phi$, and $L(t_3) \in B \cup \Phi$, where L is a literal. Suppose t_1, t_3 are unifiable with MGU $\boldsymbol{\sigma}$. Then $\boldsymbol{T}'$ constructed from $\boldsymbol{T}$ by applying $\boldsymbol{\sigma}$ to all formulas in $\boldsymbol{T}$, and extending branch $\boldsymbol{\sigma}B$ with $L(\boldsymbol{\sigma}t_2)$ is a tableau for Φ.

Closure Suppose $\boldsymbol{T}$ is a tableau for Φ and B a branch in $\boldsymbol{T}$, and L, L' are literals in $B \cup \Phi$. If $L, \overline{L'}$ are unifiable with MGU $\boldsymbol{\sigma}$ then $\boldsymbol{T}'$ constructed from $\boldsymbol{T}$ by applying σ to all formulas in $\boldsymbol{T}$ is a tableau for Φ.

Any tableau branch can be thought of as a database Φ of formulas true on that branch. Because our databases may contain (negated) identities, we need some preliminaries in order to define closure of a tableau. When checking for closure, we may consider the parameters from V occurring in literals along a tableau branch as existentially quantified. Occurrence of Pv along branch B does *not* mean that everything has property P, but rather that the thing referred to as v has P. Thus, the V-variables occurring in literals can be taken as *names*. We can *freeze* the parameters from $\boldsymbol{X}$ by mapping them to fresh parameters from V. Applying a freezing substitution to a tableau replaces references to 'arbitrary objects' $\boldsymbol{x}$, $\boldsymbol{y}$, ..., by 'arbitrary names.' What this means is that we can determine closure of a branch B in terms of the *congruence closure* of the set of equalities occurring in a frozen image $\boldsymbol{\sigma}B$ of the branch. See [5], Chapter 4, for what follows about congruence closure.

If Φ is set of $\mathcal{L}_{\Sigma^*}$ formulas without parameters from $\boldsymbol{X}$, the congruence closure of Φ, notation $\approx_\Phi$, is the smallest congruence on T that contains all the equalities in Φ. In

general, $\approx_\Phi$ will be infinite: if $a \doteq b$ is an equality in Φ, and f is a one-placed function symbol in the language, then $\approx_\Phi$ will contain $fa \doteq fb, ffa \doteq ffb, fffa \doteq fffb, \ldots$. Therefore, one uses congruence closure modulo some finite set instead.

Let S be the set of all sub-terms (not necessarily proper) of terms occurring in a literal in Φ. Then the congruence closure of Φ modulo S, notation $\mathrm{CC}_S(\Phi)$, is the finite set of equalities $\approx_\Phi \cap (S \times S)$. We can decide whether $t \doteq t'$ in $\mathrm{CC}_S(\Phi)$; [5] gives an algorithm for computing $\mathrm{CC}_S(G)$, for finite sets of equalities G and terms S, in polynomial time.

Definition 5.1 *$t \approx t'$ is suspended in frozen $\mathcal{L}_{\Sigma^*}$ formula set Φ if $t \doteq t' \in CC_S(\Phi)$, where S is the set of all sub-terms of terms occurring in literals in Φ. We extend this notation to sequences: $\bar{t} \approx \bar{t}'$ is suspended in Φ if $t_1 \approx t'_1, \ldots, t_n \approx t'_n$ are suspended in Φ.*

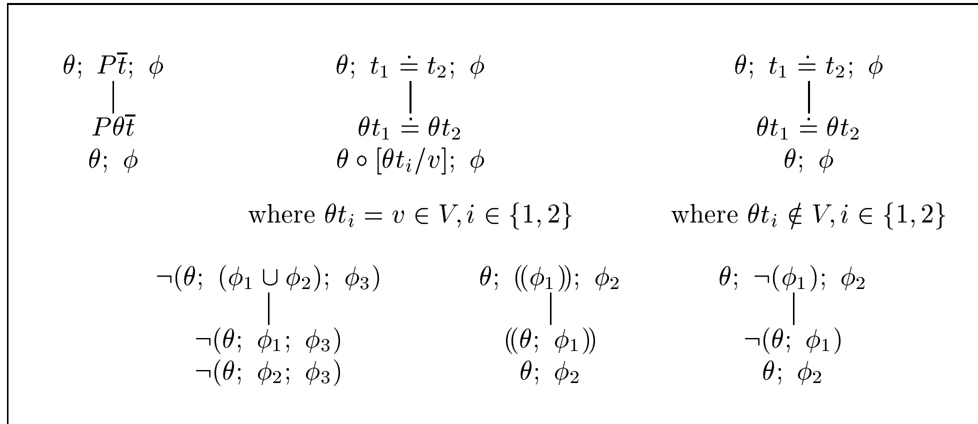
A frozen $\mathcal{L}_{\Sigma^*}$ formula set Φ is closed if either $\neg(\theta) \in \Phi$ (recall that $\perp$ is an abbreviation for $\neg([])$), or for some $\bar{t} \approx \bar{t}'$ suspended in Φ we have $P\bar{t} \in \Phi$, $\neg P\bar{t}' \in \Phi$, or for a pair of terms t_1, t_2 with $t_1 \approx t_2$ suspended in Φ we have $t_1 \neq t_2 \in \Phi$.

A tableau $\boldsymbol{T}$ is closed *if there is a freezing substitution $\boldsymbol{\sigma}$ of $\boldsymbol{T}$ such that each of its branches $\boldsymbol{\sigma}B$ is closed.*

6 Tableau Expansion Rules

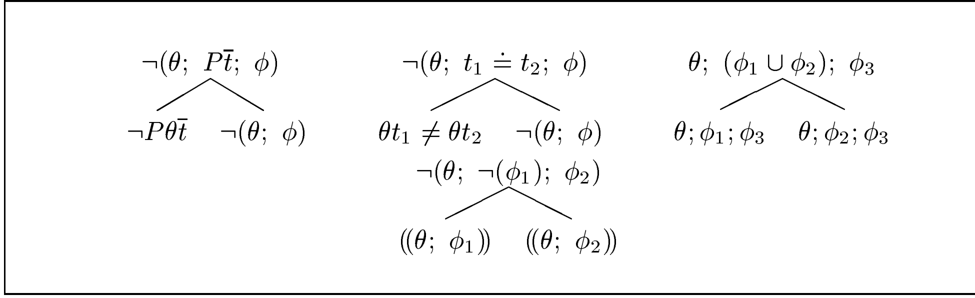
Note that we can take the form of any $\mathcal{L}_{\Sigma^*}$ formula to be $\theta; \phi$, by prefixing or suffixing [] as the need arises. The tableau rules have the effect that bindings get pushed from left to right in the tableaux, and appear as computed results at the open end nodes.

Conjunctive Type Here are the rules for formulas of conjunctive type (type α in the Smullyan taxonomy):



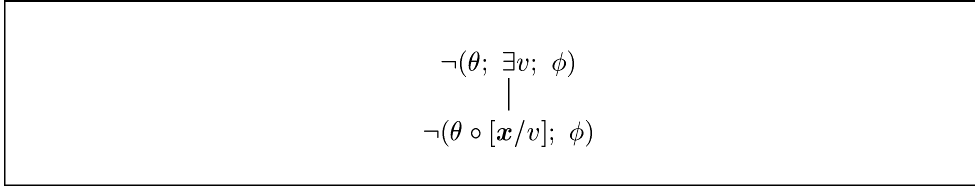
Call the formula at the top node of a rule of this kind α and the formulas at the leaves α_1, α_2. To expand a tableau branch B by an α rule, extend B with both α_1 and α_2.

Disjunctive Type The rules for formulas of disjunctive type (Smullyan's type β):



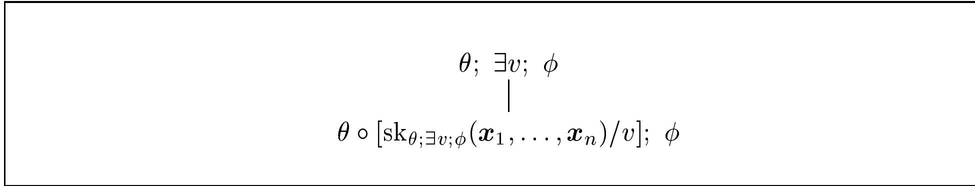
Call the formula at the top node of a rule of this kind β, the formula at the left leaf β_1 and the formula at the right leaf β_2. To expand a tableau branch B by an β rule, either extend B with β_1 or with β_2.

Universal Type Rule for universal formulas (Smullyan's type γ):



Here $\boldsymbol{x}$ is a universal variable taken from $\boldsymbol{X}$ that is new to the tableau. Call the formula at the top node of a rule of this kind $\gamma(v)$, and the formula at the leaf γ_1. To expand a tableau branch B by an γ rule, extend B with γ_1.

Existential Type Rule for existential formulas (Smullyan's type δ):



Here $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ are the universal parameters upon which interpretation of $\exists v; \phi$ depends, and $\mathrm{sk}_{\theta;\exists v;\phi}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ is a skolem constant that is new to the tableau branch.[2]

By Proposition 3.6, $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ is a subset of $input(\theta; \exists v; \phi)$, or, since no members of $\boldsymbol{X}$ occur in ϕ or in $dom(\theta)$, a subset of $\boldsymbol{X} \cap input(\theta) = \boldsymbol{X} \cap var(rng(\theta))$. From this set, we only need[3]

$$\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} := \boldsymbol{X} \cap var(rng(\theta \upharpoonright (input(\phi) \backslash \{v\}))).$$

Call the formula at the top node of a rule of this kind $\delta(v)$, and the formula at the leaf δ_1. To expand a tableau branch B by an δ rule, extend B with δ_1.

---

[2] It is well-known that this can be optimized so that the choice of skolem constant only depends on $\theta;\ \exists v;\ \phi$.

[3] In an implementation, it may be more efficient to not bother about computing $input(\phi)$, and instead work with $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} := \boldsymbol{X} \cap var(rng(\theta))$.

Protected Versions of the Rules All of the rules above have protected versions, i.e., versions with the formula ϕ to which the rule applies of the form $\psi^{\boxdot}$. The blocking operator is inherited by all the daughter formulas. As an example, here are the protected versions of one of the conjunctive and one of the disjunctive rules:

$$(\theta; P\bar{t}; \phi)^{\boxdot} \;\Big|\; (P\theta\bar{t})^{\boxdot} \quad (\theta;\phi)^{\boxdot} \qquad\qquad (\theta; (\phi_1 \cup \phi_2); \phi_3)^{\boxdot} \;\rightarrow\; (\theta; \phi_1; \phi_3)^{\boxdot} \quad (\theta; \phi_2; \phi_3)^{\boxdot}$$

Applying Definition 2.3, we see that this boils down to the following:

$$((\theta; P\bar{t}; \phi)) \;\Big|\; P\theta\bar{t} \quad ((\theta;\phi)) \qquad\qquad ((\theta; (\phi_1 \cup \phi_2); \phi_3)) \;\rightarrow\; ((\theta; \phi_1; \phi_3)) \quad ((\theta; \phi_2; \phi_3))$$

The tableau calculus specifies guidelines for extending a tableau tree with new leaf nodes. If one starts out from a single formula, at each stage only a finite number of rules can be applied. Breadth first search will get us all the possible tableau developments for a given initial formula, but this procedure is not an *algorithm* for tableau proof construction: as in the tableau systems for classical FOL, there is no guarantee of termination.

7 Soundness of the Tableau Calculus

Valuations for Σ^* models $\mathcal{M} = (D, I)$ are functions in $V \cup \boldsymbol{X} \to D$. Any such function g can be viewed as a union $s \cup h$ of a function $s \in V \to D$ and a function $h \in \boldsymbol{X} \to D$ (take $s = g \restriction V$ and $h = g \restriction \boldsymbol{X}$). For satisfaction in Σ^* models we use the notation ${}_{s\cup h}[\![\phi]\!]^{M}_{u}$, to be understood in the obvious way. In terms of this we define the notion that we need to account for the universal nature of the $\boldsymbol{X}$ variables.

Definition 7.1 *Let $\phi \in \mathcal{L}_{\Sigma^*}$, $\mathcal{M} = (D, I)$ a Σ^* model, $s, u \in V \to D$.*

Then ${}^{\forall}_{s}[\![\phi]\!]^{\mathcal{M}}$ iff for every $h : \boldsymbol{X} \to D$ there is a $u : V \cup \boldsymbol{X} \to D$ with ${}_{s\cup h}[\![\phi]\!]^{\mathcal{M}}_{u}$. We say: s universally satisfies ϕ in $\mathcal{M}$.

For any tableau $\boldsymbol{T}$ we say that $\mathbf{C}(\boldsymbol{T})$ if there is an Σ^* model $\mathcal{M}$, a branch B of $\boldsymbol{T}$ and a V valuation s for $\mathcal{M}$ such that every formula ϕ of B is universally satisfied by s in $\mathcal{M}$.

Lemma 7.2 *If s universally satisfies ϕ in $\mathcal{M}$, and $\boldsymbol{\sigma}$ is a substitution on $\boldsymbol{X}$ that is safe for ϕ, then s universally satisfies $\boldsymbol{\sigma}\phi$ in $\mathcal{M}$.*

PROOF. If ${}^{\forall}_{s}[\![\phi]\!]^{\mathcal{M}}$ then for every $\boldsymbol{X}$ valuation h in $\mathcal{M}$ there is a $V \cup \boldsymbol{X}$ valuation u in $\mathcal{M}$ with ${}_{s\cup h}[\![\phi]\!]^{\mathcal{M}}_{u}$. Thus for every h in $\mathcal{M}$ there is a $V \cup \boldsymbol{X}$ valuation u in $\mathcal{M}$ with

$${}_{s\cup h_{\boldsymbol{\sigma}}}[\![\phi]\!]^{\mathcal{M}}_{u},$$

and therefore for every h in $\mathcal{M}$ there is a $V \cup \boldsymbol{X}$ valuation u in $\mathcal{M}$ with

$${}_{s\cup h}[\![\boldsymbol{\sigma}; \phi]\!]^{\mathcal{M}}_{u}.$$

Since $\boldsymbol{\sigma}$ is safe for ϕ we have by the binding lemma that $[\![\boldsymbol{\sigma}\phi]\!]^{\mathcal{M}} = [\![\boldsymbol{\sigma}; \phi]\!]^{\mathcal{M}}$, and it follows that s universally satisfies $\boldsymbol{\sigma}\phi$ in $\mathcal{M}$. ∎

With this, we can show that the tableau building rules preserve the $\mathbf{C}(\boldsymbol{T})$ relation.

Lemma 7.3 (Tableau Expansion Lemma) *1. If tableau $\boldsymbol{T}$ for Φ yields tableau $\boldsymbol{T}'$ by an application of binding composition, then $\mathbf{C}(\boldsymbol{T})$ implies $\mathbf{C}(\boldsymbol{T}')$.*

2. If tableau $\boldsymbol{T}$ for Φ yields tableau $\boldsymbol{T}'$ by an application of a tableau expansion rule, then $\mathbf{C}(\boldsymbol{T})$ implies $\mathbf{C}(\boldsymbol{T}')$.

3. If tableau $\boldsymbol{T}$ for Φ yields tableau $\boldsymbol{T}'$ by an application of equality replacement, then $\mathbf{C}(\boldsymbol{T})$ implies $\mathbf{C}(\boldsymbol{T}')$.

4. If tableau $\boldsymbol{T}$ for Φ yields tableau $\boldsymbol{T}'$ by an application of closure, then $\mathbf{C}(\boldsymbol{T})$ implies $\mathbf{C}(\boldsymbol{T}')$.

PROOF. 1. Immediate from the fact that $\theta; \rho$ and $\theta \circ \rho$ have the same interpretation.

2. All of the α and β rules are straightforward, except perhaps for the α equality rules. The change of θ to $\theta \circ [\theta t_i/v]$, where $\theta t_j = v$ $(i, j \in \{1,2\}, i \neq j,)$ reflects the fact that $\theta t_1 \doteq \theta t_2$ gives us the information to instantiate v.

The γ rule. Assume $\neg(\theta; \exists v; \phi)$ is universally satisfied by s in $\mathcal{M}$. We may assume that θ is safe for $\exists v; \phi$. If $\boldsymbol{x} \in \boldsymbol{X}$, $\boldsymbol{x}$ fresh to the tableau, then $\theta \circ [\boldsymbol{x}/v]$ will be safe for ϕ, and $\neg(\theta \circ [\boldsymbol{x}/v]; \phi)$ will be universally satisfied by s in $\mathcal{M}$.

The δ rule. Assume s universally satisfies $\theta; \exists v; \phi$ in $\mathcal{M}$. By induction on tableau structure, $dom(\theta) \subset V$. Define a new model $\mathcal{M}'$ where $\mathrm{sk}_{\theta;\exists v;\phi}$ is interpreted as the function $f : D^n \to D$ given by $f(d_1, \ldots, d_n) :=$ some d for which ϕ succeeds in $\mathcal{M}$ for input state $s_\theta[d_1/\boldsymbol{x}_1, \ldots, d_n/\boldsymbol{x}_n, d/v]$. By the fact that s universally satisfies $\theta; \exists v; \phi$ in $\mathcal{M}$ and by the way we have picked $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$, such a d must exist. Then s will universally satisfy $\theta \circ [\mathrm{sk}_{\theta;\exists v;\phi}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)/v]; \phi$ in $\mathcal{M}'$, while universal satisfaction of other formulas on the branch is not affected by the switch from $\mathcal{M}$ to $\mathcal{M}'$.

3 and 4 follow immediately from Lemma 7.2. ∎

Theorem 7.4 (Soundness) *If $\phi, \psi \in \mathcal{L}_\Sigma$, and the tableau for $\phi; \neg(\psi)$ closes, then $\phi \models \psi$.*

PROOF. If the tableau for $\phi; \neg(\psi)$ closes, then by the Tableau Expansion Lemma, there are no $\mathcal{M}, s$ such that ${}^{\forall}_{s}[\![\phi; \neg(\psi)]\!]^{\mathcal{M}}$. Since $\phi, \psi \in \mathcal{L}_\Sigma$, there are no $\mathcal{M}, s, u$ with ${}_s[\![\phi; \neg(\psi)]\!]_u^{\mathcal{M}}$. In other words, for every Σ model $\mathcal{M}$ and every pair of variable states s, u for $\mathcal{M}$ with ${}_s[\![\phi]\!]_u^{\mathcal{M}}$ there has to be a variable state u' with ${}_u[\![\psi]\!]_{u'}^{\mathcal{M}}$. Thus, we have $\phi \models \psi$ in the sense of Definition 2.5. ∎

8 Derived Principles

Universal Quantification Immediately from the definition of $\forall v(\phi)$ we get:

$$\begin{array}{c} \theta; \forall v(\phi_1); \phi_2 \\ | \\ ((\theta \circ [\boldsymbol{x}/v]; \phi_1)) \\ \theta; \phi_2 \end{array}$$

where $\boldsymbol{x} \in \boldsymbol{X}$ new to the tableau

Blocks Detachment A sequence of blocks $\pm(\phi_1); \ldots; \pm(\phi_n)$, where $\pm(\phi_i)$ is either $((\phi_i))$ or $\neg(\phi_i)$, yields the set of its components, by a series of applications of distribution of the empty substitution over block or negation. This is useful, as the formulas $\pm(\phi_1), \ldots, \pm(\phi_n)$ can be processed in any order. In a schema:

$$\begin{array}{c} \pm(\phi_1);\ldots;\pm(\phi_n) \\ | \\ \pm(\phi_1) \\ \vdots \\ \pm(\phi_n) \end{array}$$

Negation Splitting The following rules are admissible in the calculus:

$$\frac{\neg(\phi;\neg(\psi);\chi)}{(\!(\phi;\psi)\!) \quad \neg(\phi;\chi)} \qquad \frac{\neg(\phi;(\!(\psi)\!);\chi)}{(\!(\phi;\neg(\psi))\!) \quad \neg(\phi;\chi)}$$

Negation splitting can be viewed as the DFOL guise of a well known principle from modal logic: $\Box(A \vee B) \rightarrow (\Diamond A \vee \Box B)$. To see the connection, note that $\neg(\phi;\neg(\psi);\chi)$ is semantically equivalent to $\neg(\phi;\neg(\psi \cup \neg(\chi)))$, where $\neg(\phi;\neg\cdots)$ behaves as a $\Box$ modality.

9 Examples

In the examples we will use $v_0, v_1, \ldots$ as 0-ary skolem terms for v, etcetera.

Syllogistic Reasoning Consider the syllogism:

$$\forall x(Ax \rightarrow Bx), \forall x(Bx \rightarrow Cx) \models \forall x(Ax \rightarrow Cx).$$

This is an abbreviation of (9.1).

$$\neg(\exists x; Ax; \neg Bx), \neg(\exists x; Bx; \neg Cx) \ \models\ \neg(\exists x; Ax; \neg Cx) \tag{9.1}$$

The DFOL tableau for this example, a tableau refutation of

$$\neg(\exists x; Ax; \neg Bx); \neg(\exists x; Bx; \neg Cx); (\!(\exists x; Ax; \neg Cx)\!)$$

is in Figure 1.

Dynamic Donkey Reasoning The hackneyed example for dynamic binding in natural language, *If a farmer owns a donkey, he beats it*, has the following DFOL shape:

$$(\exists x; \exists y; Fx; Dy; Oxy \rightarrow Bxy),$$

which is shorthand for:

$$\neg(\exists x; \exists y; Fx; Dy; Oxy; \neg Bxy).$$

Consider the natural language text in (9.2).

If a farmer owns a donkey, he beats it. *A.* is a farmer and owns a donkey. (9.2)

Figure 2 shows how to draw conclusions from the DFOL version of this text in a DFOL tableau calculation.

The open tableau branch in Figure 2 yields the fact Baz_1, plus the following further information about z_1: Dz_1, Oaz_1. This further information is useful to identify z_1 as

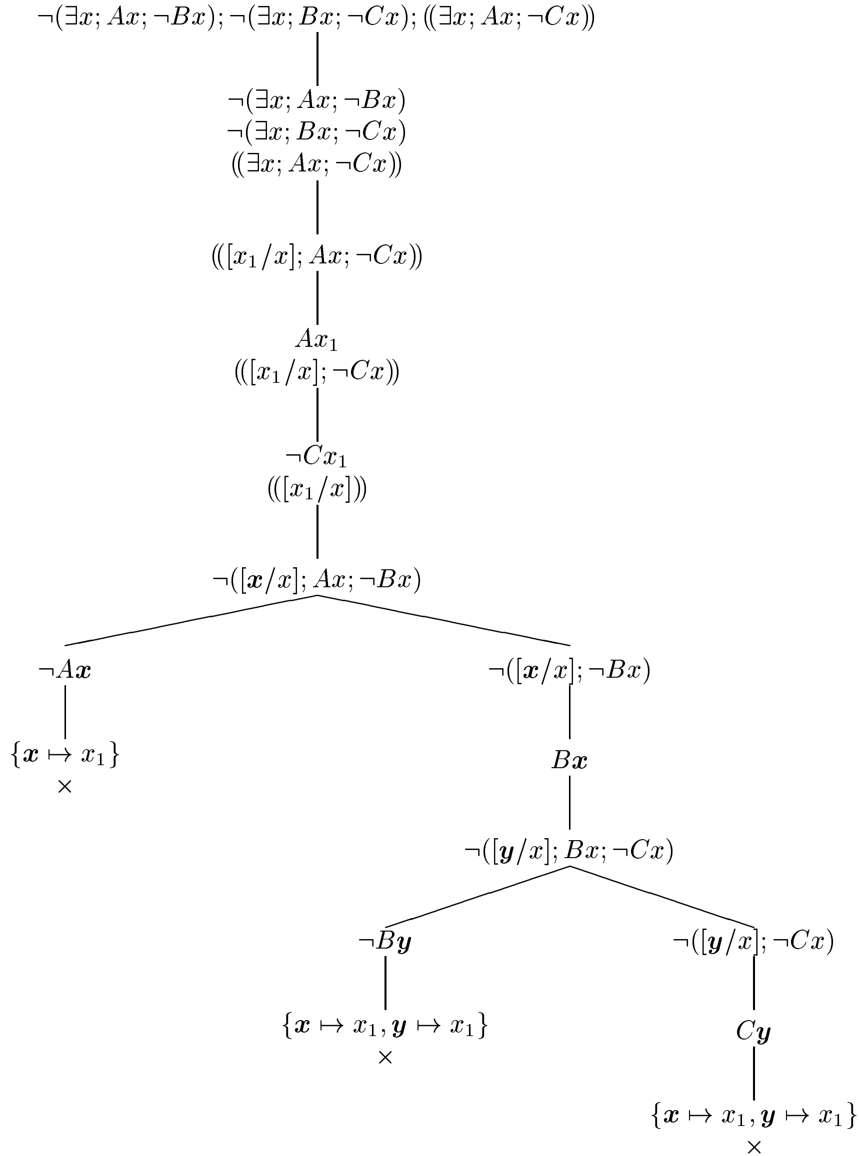


FIG. 1. DFOL Tableau for Syllogistic Reasoning (9.1)

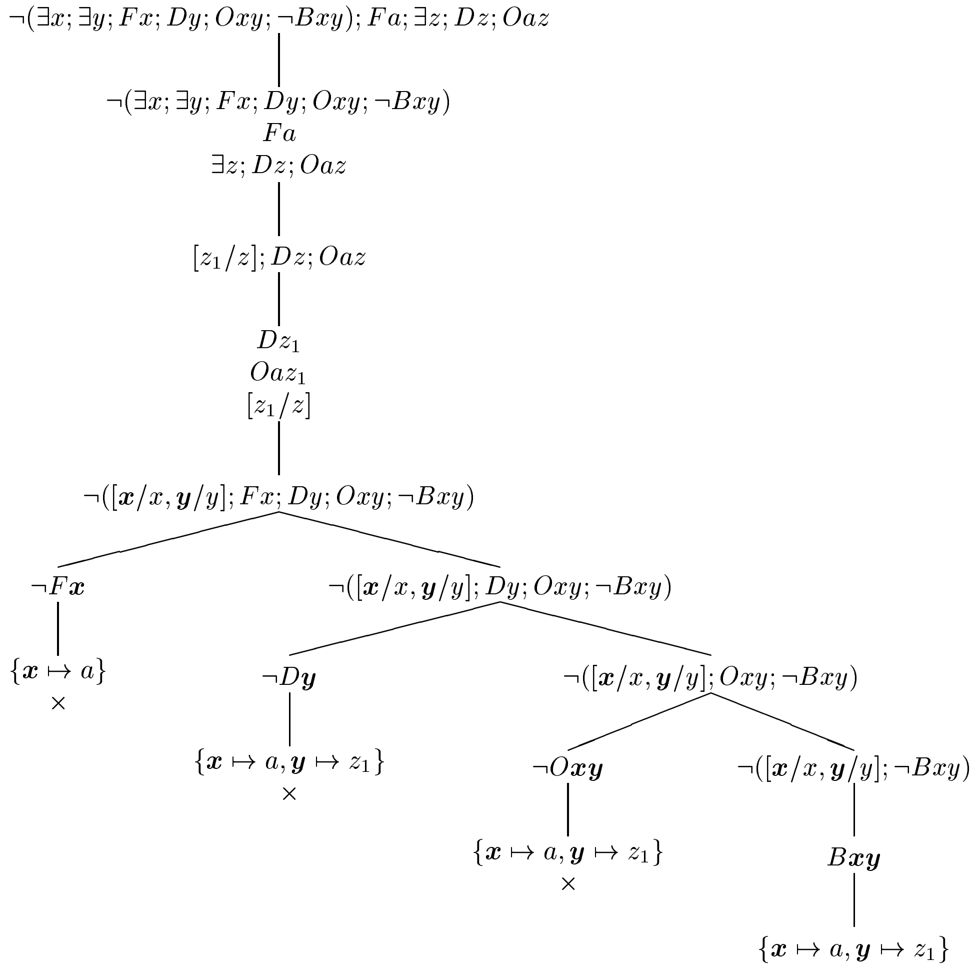


FIG. 2. Tableau for Dynamic Donkey Reasoning (9.2)

the donkey that Alfonso owns (or perhaps *a donkey that Alfonso owns*) that was introduced in the text.

Open Tableau Branches, Partial Models, Reference Resolution An open tableau branch for a DFOL formula ϕ may be viewed as a *partial model* for ϕ, with just enough information to verify the formula. For instance, the open branch in the previous example does not specify whether donkey z_1 also beats Alfonso or not: Bz_1a is neither among the facts (true atoms) nor among the negated facts (false atoms) of the branch.

In tableau branches involving equality there is also another kind of partiality involved: the terms are *proto-objects* rather than genuine objects, in sense that they have not yet 'made up their minds' about which individual they are: two terms t_1, t_2 on a tableau that does not contain $t_1 \neq t_2$ may be interpreted as a single individual. This is because the information about equality that the branch provides is also

partial. Also, variables from $\boldsymbol{X}$ (free tableau variables) can be resolved to any object whatsoever.

The level of tableau style generation of partial models for discourse may be just the right level for pronoun reference resolution (cf. the suggestion in [7]). Since reference resolution is a processing step that links a pronoun to a suitable antecedent, what about equating the suitable antecedents with the available terms of the branches in a tableau? After all, reference resolution for pronouns is part of semantic processing, so it has a more natural habitat at the level of processing NL representations than at the level of mere representation of NL meaning.

Building on this idea, we (tentatively) introduce the following rule for pronoun resolution:

$$\begin{array}{c} P\text{pro} \\ | \\ Pt \end{array} \qquad\qquad \begin{array}{c} \neg P\text{pro} \\ | \\ \neg Pt \end{array}$$

t occurs on the branch t occurs on the branch

Of course, for a full account one would need rules to determine the *salient* terms for pronoun resolution along a branch, but here we will just demonstrate the rule with a tableau for the following piece of discourse.

Every farmer owns a donkey. Some farmer beats it. (9.3)

See Figure 3. Intuitively, in this tableau, the following happens. First, a term z_1 introduced for *Some farmer*. This leads to an unresolved fact '$B(z_1, \text{it})$' in the database of the partial model under construction. Later, the pronoun *it* is resolved to 'the donkey that z_1 owns' generated from *every farmer owns a donkey*, and represented in the database of the partial model as $\text{sk}_1(z_1)$.

Here is another well-known example from the literature that is hard to crack in a purely representational setting (a piece of evidence against the claim, by the way, that 'or' in natural language is externally static):

John owns a motorbike or a car. It is in the garage. (9.4)

Again, in the tableau setting there is no problem: the tableau for (9.4) will have two branches, and both of the branches will contain a suitable antecedent for *it*.

Reasoning about '<' Consider example (9.5).

$$y < x; \neg(\exists x; \exists y; x < y). \tag{9.5}$$

This is contradictory, for first two objects of different size are introduced, and next we are told that all objects have the same size. The contradiction is derived as follows:

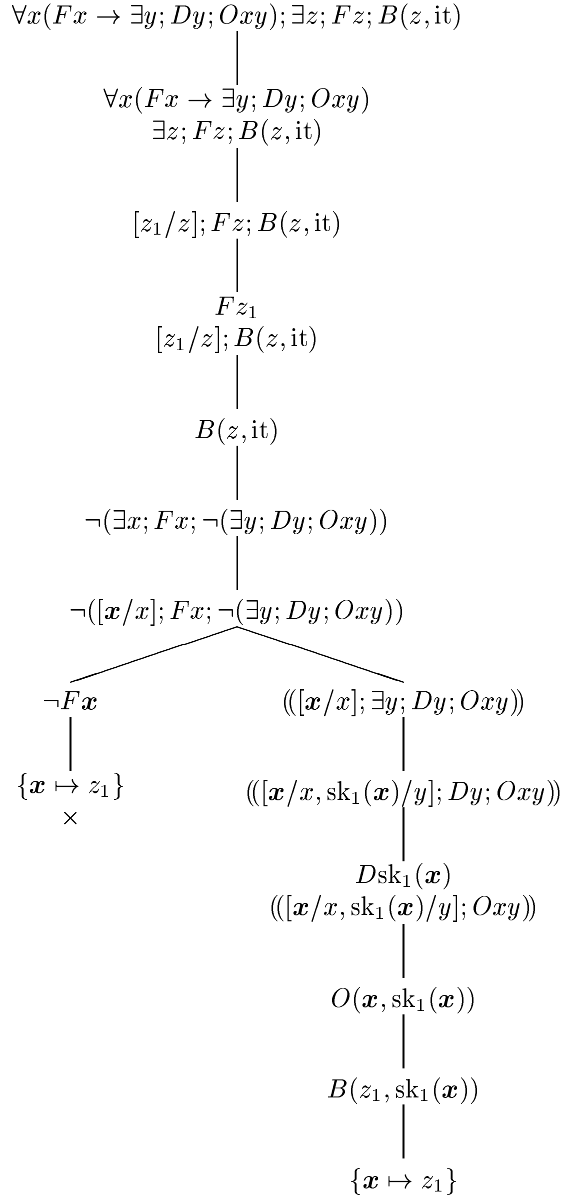


FIG. 3. Tableau for Donkey Reasoning with Pronoun Resolution (9.3)

$$y < x; \neg(\exists x; \exists y; x < y)$$
$$|$$
$$y < x$$
$$\neg(\exists x; \exists y; x < y)$$
$$|$$
$$\neg([\boldsymbol{x}_1/x, \boldsymbol{x}_2/y]; x < y)$$
$$|$$
$$\neg \boldsymbol{x}_1 < \boldsymbol{x}_2$$
$$|$$
$$\{\boldsymbol{x}_1 \mapsto y, \boldsymbol{x}_2 \mapsto x\}$$
$$\times$$

More Reasoning about $<$ Assume that $1, 2, 3, \ldots$ are shorthand for $s0, ss0, sss0, \ldots$. We derive a contradiction from the assumption that $4 < 2$ together with two axioms for $<$. See Figure 4, with arrows connecting the literals that effect closure.

Computation of Answer Substitutions, with Variable Reuse Figure 5 demonstrates how the computed answer substitution stores the final value for x, under the renaming x_1. Because of the renaming, the database information for x_1 does not conflict with that for x.

Closure by Equality Replacement This example illustrates closure by means of equality replacement, in reasoning about $\exists x; \exists y; x \neq y; \exists x; \neg(\exists y; x \neq y)$. Note that x_1, y_1, x_2 serve as names for objects in the domain under construction. What the argument boils down to is: if the name x_2 applies to everything, then it cannot be the case that there are two different objects x_1, y_1. See Figure 6.

The first application of equality replacement in Figure 6 unifies $\boldsymbol{x}$ with x_1 and concludes from $x_2 \doteq \boldsymbol{x}, x_1 \neq y_1$ that $x_2 \neq y_1$. The second application of equality replacement unifies $\boldsymbol{y}$ with y_1 and concludes from $x_2 \doteq \boldsymbol{y}, x_2 \neq y_1$ that $x_2 \neq x_2$.

Loop Invariant Checking To check that $x = y!$ is a loop invariant for $y := y + 1; x := x * y$, assume it is not, and use the calculus to derive a contradiction with the definition of !. Note that $y := y + 1; x := x * y$ appears in our notation as $[y + 1/y]; [x * y/x]$. See Figure 7. A more detailed account would of course have to use the DFOL definitions of $+$, $*$ and !.

Loop Invariant Detection This time, we inspect the code $[x * (y + 1)/x]; [y + 1/y]$ starting from scratch. Since y is the variable that gets incremented, we may assume that x depends on y via an unknown function f. Thus, we start in a situation where $fy = x$. We check what has happened to this dependency after execution of the code $[x * (y + 1)/x]; [y + 1/y]$, by means of a tableau calculation for $fy \doteq x; [x * (y + 1)/x]; [y + 1/y]; fy \doteq x$. See Figure 7. The tableau shows that $[x * (y + 1)/x]; [y + 1/y]$ is a loop for the factorial function.

Postcondition Reasoning for 'If Then Else' For another example of this, consider a loop through the following programming code:

$$i := i + 1; \text{if } x < a[i] \text{ then } x := a[i] \text{ else skip}. \tag{9.6}$$

Assume we know that before the loop x is the maximum of array elements $a[0]$ through $a[i]$. Then our calculus allows us to derive a characterization of the value of x at the end of the loop. Note that the loop code appears in DFOL under the following guise:

$$[i + 1/i]; (x < a[i]; [a[i]/x] \cup \neg x < a[i]).$$

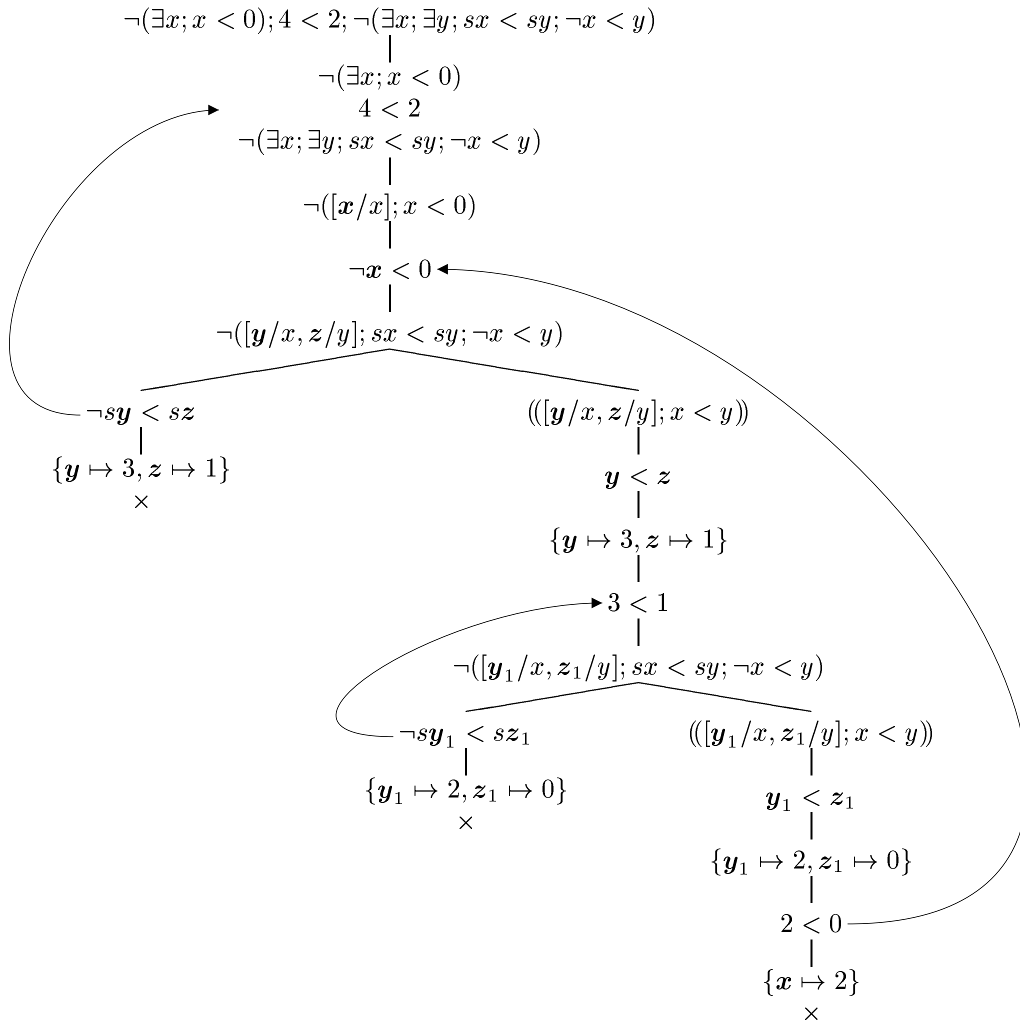


FIG. 4. More Reasoning about $<$.

The situation of x at the start of the loop can be given by an identity $x = m_i^0$, where m is a two-placed function. To get a characterization of x at the end, we just put $X = x$ (X a constant) at the end, and see what we get (Figure 8). What the leaf nodes tell us is that in any case, X is the maximum of $a[0], .., a[i+1]$, and this maximum gets computed in x.

10 Completeness

Completeness for this calculus can be proved by a variation on completeness proofs for tableau calculi in classical FOL. First we define *trace sets* for DFOL as an analogue to Hintikka sets for FOL. A trace set is a set of DFOL formulas satisfying the closure

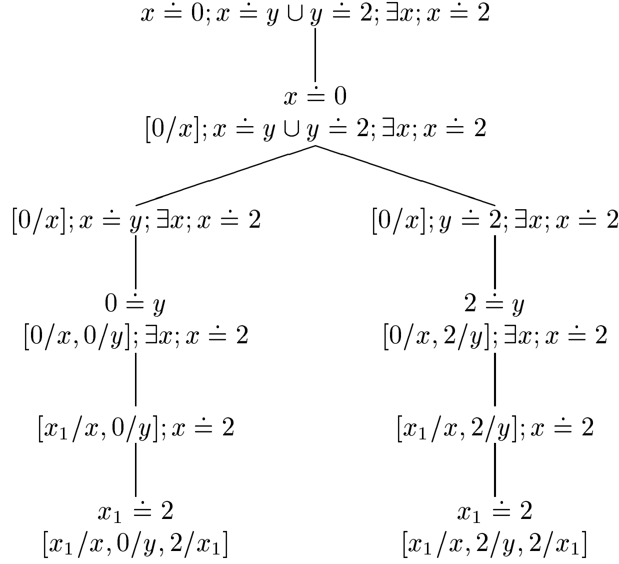


FIG. 5. Computation of Answer Substitutions, with Variable Reuse

conditions that can be read off from the tableau rules. Trace sets can be viewed as blow-by-blow accounts of particular consistent DFOL computation paths (i.e., paths that do not close).

Definition 10.1 *A set Ψ of $\mathcal{L}_{\Sigma^*}$ formulas is a* **trace set** *if the following hold:*

1. $\neg(\theta) \notin \Psi$.
2. *If $\phi \in \Psi$, then $\overline{\phi} \notin \Psi$.*
3. *If $\theta; \phi \in \Psi$, then $\theta\phi \in \Psi$.*
4. *If $\alpha \in \Psi$ then all $\alpha_i \in \Psi$.*
5. *If $\beta \in \Psi$ then at least one $\beta_i \in \Psi$.*
6. *If $\gamma(v) \in \Psi$, then $\gamma_1(t) \in \Psi$ for all $t \in T^V_{\Sigma^*}$ (all terms that do not contain variables from $\boldsymbol{X}$).*
7. *If $\delta(v) \in \Psi$, then $\delta_1(t) \in \Psi$ for some $t \in T^V_{\Sigma^*}$ (some term t that does not contain variables from $\boldsymbol{X}$).*

This definition is motivated by the Trace Lemma:

Lemma 10.2 (Trace Lemma) *The elements of every trace set Ψ are simultaneously satisfiable.*

PROOF. Define a canonical model $\mathcal{M}_0$ in the standard fashion, using congruence closure on the trace set Ψ over the set of terms occurring in Φ, to get a suitable congruence $\equiv$ on terms. Next, define a canonical valuation s_0 by means of $s_0(v) := [v]_\equiv$ for members of V and $s_0(\mathrm{sk}^0_i) = [\mathrm{sk}^0_i]_\equiv$ for 0-ary skolem terms. Verify that s_o satisfies every member of Φ in $\mathcal{M}_0$. ∎

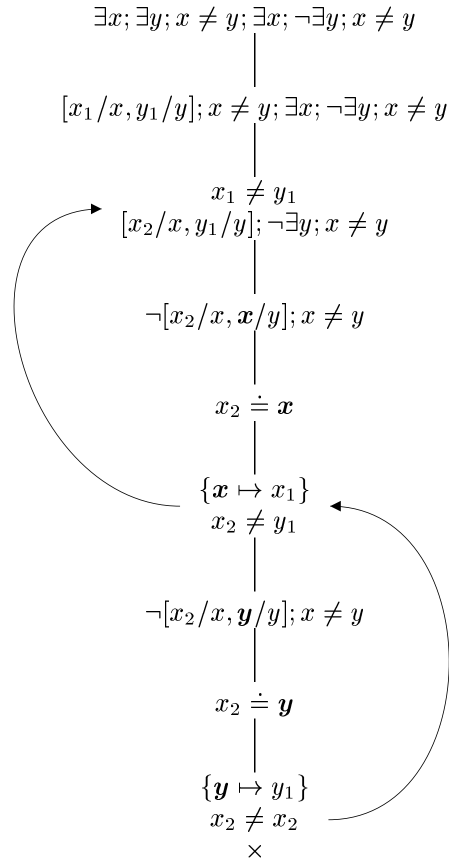


FIG. 6. Reasoning With Equality

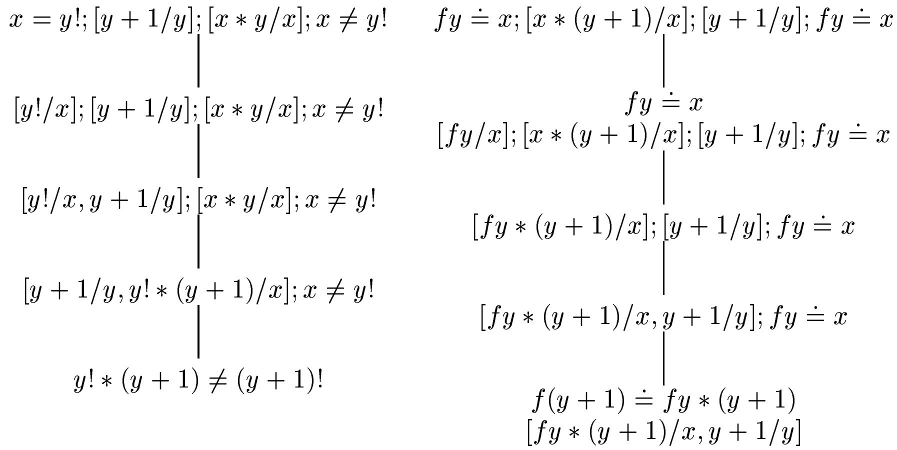


FIG. 7. Loop Invariant Checking and Loop Invariant Detection.

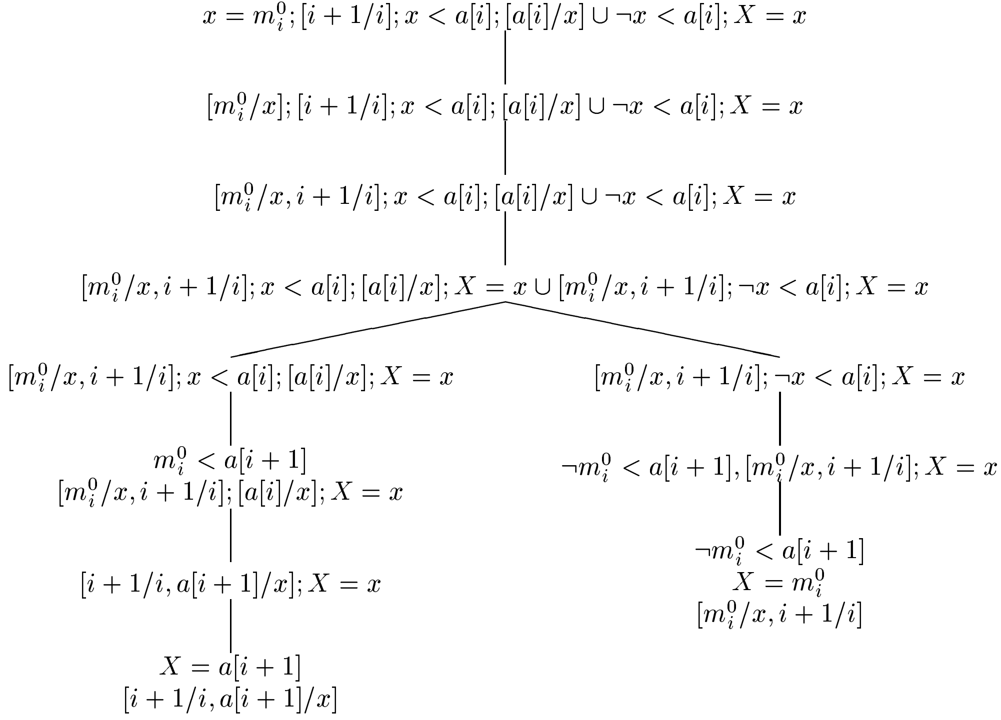


FIG. 8. Postcondition Reasoning for (9.6)

To employ the lemma, we need the standard notion of a fair computation rule. A computation rule is a function F that for any set of formulas Φ and any tableau $\boldsymbol{T}$, computes the next rule to be applied on $\boldsymbol{T}$. This defines a partial order on the set of tableaux for Φ, with the successor of $\boldsymbol{T}$ given by F. Then there is a (possibly infinite) sequence of tableaux for Φ starting from the initial tableau, and with supremum $\boldsymbol{T}_\infty$. A computation rule F is fair if the following holds for all branches B in $\boldsymbol{T}_\infty$:

1. All formulas of type α, β, δ occurring on B or in Φ were used to expand B,
2. All formulas of type γ occurring on B or in Φ were used infinitely often to expand B.

Theorem 10.3 (Completeness) *For all $\phi, \psi \in \mathcal{L}_\Sigma$: if $\phi \models \psi$ then there is a tableau refutation of $\phi; \neg(\psi)$.*

PROOF. Let $\boldsymbol{T}_0, \ldots$ be a sequence of tableaux for $\phi; \neg(\psi)$ constructed with a fair computation rule, without closure rule applications, and with supremum $\boldsymbol{T}_\infty$. Define a freezing map $\boldsymbol{\sigma}_\infty$ on $\boldsymbol{T}_\infty$ as follows (see, e.g., [17]). Let $(B_k)_{k \geq 0}$ be an enumeration of the branches of $\boldsymbol{T}_\infty$, let $(\phi_i)_{i \geq 0}$ be an enumeration of the type γ formulas of $\boldsymbol{T}_\infty$, and let $\boldsymbol{x}_{ijk}$ be the variable introduced for the j-th application of γ formula ϕ_i along branch B_k. If $(t_j)_{j \geq 0}$ is an enumeration of all the frozen terms of $\boldsymbol{T}_\infty$, we can set $\boldsymbol{\sigma}_\infty(\boldsymbol{x}_{ijk}) := t_j$ for all $i, j, k \geq 0$. Note that $\boldsymbol{\sigma}_\infty$ is not, strictly speaking, a substitution since $dom(\boldsymbol{\sigma}_\infty)$ is not finite.

Suppose $\boldsymbol{\sigma}_\infty \boldsymbol{T}_\infty$ contains an open branch. Then from this branch we get a trace set, which in turn would give a canonical model and a canonical valuation for $\phi; \neg(\psi)$, and contradiction with the assumption that $\phi \models \psi$. Therefore, $\boldsymbol{\sigma}_\infty \boldsymbol{T}_\infty$ must be closed.

Since the tree $\boldsymbol{T}_\infty$ is finitely branching and all formulas having an effect on closure are at finite distance from the root, there is a finite $\boldsymbol{T}_n$ with $\boldsymbol{\sigma}_\infty \boldsymbol{T}_n$ closed. Finally, construct an MGU $\boldsymbol{\sigma}$ for $\boldsymbol{T}_n$ on the basis of the part of $\boldsymbol{\sigma}_\infty$ that is actually used in the closure of $\boldsymbol{T}_n$, and we are done. ∎

Theorem 10.4 (Computation Theorem) *If ϕ is satisfiable, then all bindings θ produced by open tableau branches B satisfy ${}_s[\![\phi]\!]^{\mathcal{M}}_{s_\theta}$, where $\mathcal{M}$ is the canonical model constructed from B, and s the canonical valuation.*

PROOF. Let $\boldsymbol{T}_0, \ldots$ be a sequence of tableaux for ϕ constructed with a fair computation rule, without closure rule applications, and with supremum $\boldsymbol{T}_\infty$. Consider $\boldsymbol{\sigma}_\infty \boldsymbol{T}_\infty$, where $\boldsymbol{\sigma}_\infty$ is the canonical freezing substitution. Then since ϕ is satisfiable, $\boldsymbol{\sigma}_\infty \boldsymbol{T}_\infty$ will have open branches $(B_k)_{k \geq 0}$ (the number need not be finite). It follows from the format of the tableau expansion rules that every open branch will develop one binding.

We say that a binding θ occurs non-protected in a formula ϕ if ϕ has the form $\theta; \psi$. Check that the tableau expansion rules on formulas of the forms $(\!(\psi)\!)$ or $\neg(\psi)$ never yield non-protected bindings $\theta \neq []$. Check that each application of an α, β, γ or δ rule to a formula with a non-protected binding extends a branch with exactly one non-protected binding. It follows that every tableau branch B_k has a highest node where a formula of the form θ appears. This θ can be thought of as the result of pulling the initial binding $[]$ through the initial formula ϕ. For every such B_k and θ there is a finite $\boldsymbol{T}_n$ with a branch $B_{k'}$ that already contains (a generalization of) θ.

It can be proved by induction on the length of $B_{k'}$ that ${}_s[\![\phi]\!]^{\mathcal{M}}_{s_\theta}$, for $\mathcal{M}$ the canonical model and s the canonical valuation for that branch. ∎

Note that the computation theorem gives no recipe for generating all correct bindings for a given ϕ. Specifying appropriate computation rules for generating these bindings for specific sets of DFOL formulas remains a topic for future research.

Variation: Using the Calculus with a Fixed Model Computing with respect to a fixed model is but a slight variation on the general scheme. The technique of using tableau rules for model checking is well known. Assume that a model $\mathcal{M} = (D, I)$ is given. Then instead of storing ground predicates $P\theta\bar{t}$ (ground equalities $\theta t_1 \doteq \theta t_2$), we check the model for $\mathcal{M} \models P\theta\bar{t}$ (for $[\![\theta t_1]\!]^{\mathcal{M}} = [\![\theta t_2]\!]^{\mathcal{M}}$), and close the branch if the test fails, continue otherwise. Similarly, instead of storing ground predicates $P\theta\bar{t}$ (ground equalities $\theta t_1 \doteq \theta t_2$) under negation, we check the model for $\mathcal{M} \not\models P\theta\bar{t}$ (for $[\![\theta t_1]\!]^{\mathcal{M}} \neq [\![\theta t_2]\!]^{\mathcal{M}}$), and close the branch if the test fails, continue otherwise.

11 Adding Iteration

Let $\mathcal{L}^*_\Sigma$ be the language that results from extending $\mathcal{L}_\Sigma$ with formulas of the form ϕ^*. The intended relational meaning of ϕ^* is that ϕ gets executed a finite (≥ 0) number of times. This extension makes $\mathcal{L}^*_\Sigma$ into a full-fledged programming language, with its assertion language built in for good measure.

The semantic clause for ϕ^* runs as follows:

$$_s[\![\phi^*]\!]_u^{\mathcal{M}} \quad \text{iff} \quad \text{either } s = u$$
$$\text{or } \exists s_1, \ldots, s_n (n \geq 1) \text{ with } {}_s[\![\phi]\!]_{s_1}^{\mathcal{M}}, \ldots, {}_{s_n}[\![\phi]\!]_u^{\mathcal{M}}.$$

It is easy to see that it follows from this definition that:

$$_s[\![\phi^*]\!]_u^{\mathcal{M}} \text{ iff either } s = u \text{ or } \exists s_1 \text{ with } {}_s[\![\phi]\!]_{s_1}^{\mathcal{M}} \text{ and } {}_{s_1}[\![\phi^*]\!]_u^{\mathcal{M}}. \qquad (11.1)$$

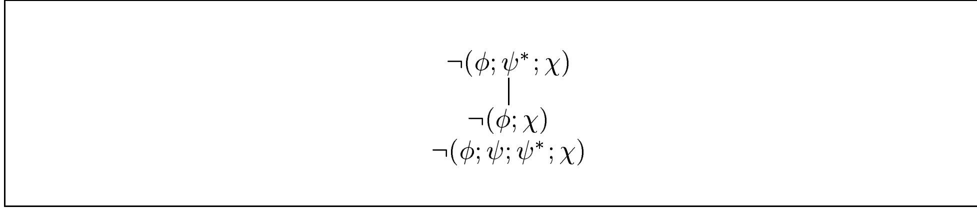
Note, however, that (11.1) is not equivalent to the definition of ${}_s[\![\phi^*]\!]_u^{\mathcal{M}}$, for (11.1) does not rule out infinite ϕ paths.

Let ϕ^n be given by: $\phi^0 := []$ and $\phi^{n+1} := \phi; \phi^n$. Now ϕ^* is equivalent to 'for some $n \in \mathbb{N} : \phi^n$'.

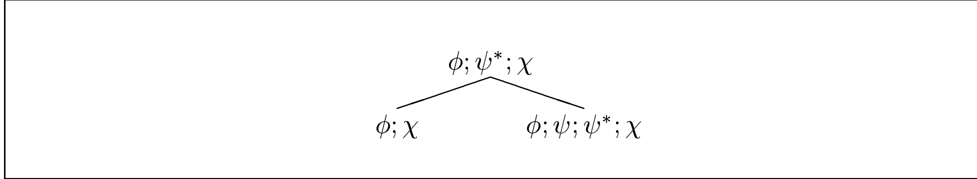
What we will do in our calculus for DFOL* is take (11.1) as the cue to the star rules. This will allow star computations to loop, which does not pose any problem, given that we extend our notion of closure to 'closure in the limit' (see below).

The calculus for DFOL* has all expansion rules of the DFOL calculus, plus the following α^* and β^* rules.

α^* *expansion rule* Call ψ^* the star formula of the rule.



β^* *expansion rule* Call ψ^* the star formula of the rule. The β^* rule also has a protected version.



To see that the α^* rule is sound, assume that s universally satisfies $\neg(\phi; \psi^*; \chi)$ in $\mathcal{M} = (D, I)$. By (11.1), this means that there is at least one $h : \boldsymbol{X} \to D$ for which there is no u with ${}_{s \cup h}[\![\phi; \chi]\!]_u^{\mathcal{M}}$ and no u with ${}_{s \cup h}[\![\phi; \psi; \psi^*; \chi]\!]_u^{\mathcal{M}}$. Thus, s universally satisfies $\neg(\phi; \chi)$ and $\neg(\phi; \psi; \psi^*; \chi)$ in $\mathcal{M}$.

For the β^* rule, assume that s universally satisfies $\phi; \psi^*; \chi$ in $\mathcal{M}$. Then for every $h : \boldsymbol{X} \to D$ there are u, u' with ${}_{s \cup h}[\![\phi]\!]_u^{\mathcal{M}}$ and ${}_u[\![\psi^*; \chi]\!]_{u'}^{\mathcal{M}}$. Then, by (11.1), either ${}_u[\![\chi]\!]_{u'}^{\mathcal{M}}$ or there is a u_1 with ${}_u[\![\psi]\!]_{u_1}^{\mathcal{M}}$ and ${}_{u_1}[\![\phi_1^*; \chi]\!]_{u'}^{\mathcal{M}}$. Thus, s universally satisfies either $\phi; \chi$ or $\phi; \psi; \psi^*; \chi$ in $\mathcal{M}$.

Closure in the Limit To deal with the inflationary nature of the α^* and β^* rules (the star formula of the rule reappears at a leaf node), we need a modification of our notion of tableau closure. We allow closure in the limit, as follows.

Definition 11.1 *An infinite tableau branch closes in the limit if it contains an infinite star development, i.e., an infinite number of α^* or β^* applications to the same star formula.*

Example of Closure in the Limit We will give an example of an infinite star development. Consider formula (11.2):

$$\neg\exists w\neg(\exists v; v = 0; (v \neq w; [v + 1/v])^*; v = w). \tag{11.2}$$

What (11.2) says is that there is no object w that cannot be reached in a finite number of steps from $v = 0$, or in other words that the successor relation $v \mapsto v+1$, considered as a graph, is well-founded. This is the Peano induction axiom: it characterizes the natural numbers up to isomorphism. What it says is that any set A that contains 0 and is closed under successor contains all the natural numbers. The fact that Peano induction is expressible as an $\mathcal{L}_\Sigma^*$ formula is evidence that $\mathcal{L}_\Sigma^*$ has greater expressive power than FOL. In FOL no single formula can express Peano induction: no formula can distinguish the standard model $(\mathbb{N}, s)$ from the non-standard models. In a non-standard model of the natural numbers it may take an infinite number of s-steps to get from one natural number n to a larger number m.

The expressive power of $\mathcal{L}_\Sigma{}^*$ is the same as that of quantified dynamic logic ([25, 15]). Arithmetical truth is undecidable, so there can be no finitary refutation system for $\mathcal{L}_\Sigma^*$. The finitary tableau system for $\mathcal{L}_\Sigma$ is evidence for the fact that DFOL validity is recursively enumerable: all non-validities are detected by a finite tableau refutation. This property is lost in the case of $\mathcal{L}_\Sigma^*$: the language is just too expressive to admit of finitary tableau refutations.

Therefore, some tableau refutations must be infinitary, and the tableau development for the negation of (11.2) is a case in point. Let us see what happens if we attempt to refute the negation of (11.2). A successful refutation will identify the natural numbers up to isomorphism. See Figure 9. This is indeed a successful refutation, for the tree closes in the limit. But the refutation tree is infinite: it takes an infinite amount of time to do all the checks.

Theorem 11.2 (Soundness Theorem for $\mathcal{L}_\Sigma^*$) *The calculus for DFOL* is sound:*

For all $\phi, \psi \in \mathcal{L}_\Sigma^*$: if the tableau for $\phi; \neg(\psi)$ closes then $\phi \models \psi$.

The modified tableau method does not always give finite refutations. Still, it is a very useful reasoning tool, more powerful than Hoare reasoning, and more practical than the infinitary calculus for quantified dynamic logic developed in [14, 15]. Dynamic logic itself has been put to practical use, e.g. in KIV, a system for interactive software verification [26]. It is our hope that the present calculus can be used to further automate the software verification process.

Precondition/postcondition Reasoning For a further example of reasoning with the calculus, consider formula (11.3). This gives an $\mathcal{L}_\Sigma^*$ version of Euclid's GCD algorithm.

$$(x \neq y; (x > y; [x - y/x] \cup y > x; [y - x/y]))^*; x \doteq y. \tag{11.3}$$

To do automated precondition-postcondition reasoning on this, we must find a trivial correctness statement. Even if we don't know what $\gcd(x,y)$ is, we know that its value should not change during the program. So putting $\gcd(x,y)$ equal to some

$$\exists w \neg(\exists v; v \doteq 0; (v \neq w; [v+1/v])^*; v \doteq w)$$

$$|$$

$$[w_1/w]\neg(\exists v; v \doteq 0; (v \neq w; [v+1/v])^*; v \doteq w)$$

$$|$$

$$\neg([w_1/w, 0/v]; (v \neq w; [v+1/v])^*; v \doteq w)$$

$$|$$

$$\neg([w_1/w, 0/v]; v \doteq w)$$

$$\neg([w_1/w, 0/v]; v \neq w; [v+1/v]; (v \neq w; [v+1/v])^*; v \doteq w)$$

$$|$$

$$0 \neq w_1$$

$$|$$

$$\neg([w_1/w, 1/v]; (v \neq w; [v+1/v])^*; v \doteq w)$$

$$|$$

$$\neg([w_1/w, 1/v]; v \doteq w)$$

$$\neg([w_1/w, 1/v]; v \neq w; [v+1/v]; (v \neq w; [v+1/v])^*; v \doteq w)$$

$$|$$

$$1 \neq w_1$$

$$|$$

$$\neg([w_1/w, 2/v]; (v \neq w; [v+1/v])^*; v \doteq w)$$

$$|$$

$$\neg([w_1/w, 2/v]; v \doteq w)$$

$$\neg([w_1/w, 2/v]; v \neq w; [v+1/v]; (v \neq w; [v+1/v])^*; v \doteq w)$$

$$|$$

$$2 \neq w_1$$

$$|$$

$$\neg([w_1/w, 3/v]; (v \neq w; [v+1/v])^*; v \doteq w)$$

$$|$$

$$\neg([w_1/w, 3/v]; v \doteq w)$$

$$\neg([w_1/w, 3/v]; v \neq w; [v+1/v]; (v \neq w; [v+1/v])^*; v \doteq w)$$

$$|$$

$$3 \neq w_1$$

$$|$$

$$\neg([w_1/w, 4/v]; (v \neq w; [v+1/v])^*; v \doteq w)$$

$$|$$

$$\neg([w_1/w, 4/v]; v \doteq w)$$

$$\neg([w_1/w, 4/v]; v \neq w; [v+1/v]; (v \neq w; [v+1/v])^*; v \doteq w)$$

$$|$$

$$4 \neq w_1$$

$$|$$

$$\neg([w_1/w, 5/v]; (v \neq w; [v+1/v])^*; v \doteq w)$$

$$|$$

$$\vdots$$

$$\times$$

FIG. 9. 'Infinite Proof' of the Peano Induction Axiom.

arbitrary value and see what happens would seem to be a good start. We will use the correctness statement $z \doteq \text{gcd}(x, y)$. The statement that the result gets computed in x can then take the form $z \doteq x$. The program with these trivial correctness statements included becomes:

$$
\begin{aligned}
& z \doteq \text{gcd}(x, y); \\
& (x \neq y; (x > y; [x - y/x]; z \doteq \text{gcd}(x, y) \cup y > x; [y - x/y]; z \doteq \text{gcd}(x, y)))^*; \\
& x \doteq y; z \doteq x.
\end{aligned} \tag{11.4}
$$

We can now put the calculus to work. Abbreviating

$$(x \neq y; (x > y; [x - y/x]; z \doteq \text{gcd}(x, y) \cup y > x; [y - x/y]; z \doteq \text{gcd}(x, y)))^*$$

as A^*, we get:

$$z \doteq \text{gcd}(x, y); A^*; x \doteq y; z \doteq x$$

Left branch: $[\text{gcd}(x, y)/z]; x \doteq y; z \doteq x$, followed by $x \doteq y, \text{gcd}(x, y) \doteq x$

Right branch: $[\text{gcd}(x, y)/z]; A; A^*; x \doteq y; z \doteq x$, which splits into:

- $x > y$; $\text{gcd}(x, y) \doteq \text{gcd}(x - y, y)$; $[\text{gcd}(x, y)/z, x - y/x]; A^*;$ $x \doteq y; z \doteq x$
- $y > x$; $\text{gcd}(x, y) \doteq \text{gcd}(x, y - x)$; $[\text{gcd}(x, y)/z, y - x/y]; A^*;$ $x \doteq y; z \doteq x$

The second split is caused by an application of the rule for $\cup$. By the soundness of the calculus any model satisfying the annotated program (11.4) will satisfy one of the branches. This shows that if the program succeeds (computes an answer), the following disjunction will be true:

$$
\begin{aligned}
& (x \doteq y \wedge \text{gcd}(x, y) \doteq x) \\
\vee \ \ & (x > y \wedge \text{gcd}(x, y) \doteq \text{gcd}(x - y, y) \wedge \phi) \\
\vee \ \ & (y > x \wedge \text{gcd}(x, y) \doteq \text{gcd}(x, y - x) \wedge \psi),
\end{aligned} \tag{11.5}
$$

where ϕ and ψ abbreviate, respectively, $[\text{gcd}(x, y)/z, x - y/x]; A^*; x \doteq y; z \doteq x$ and $[\text{gcd}(x, y)/z, y - x/y]; A^*; x \doteq y; z \doteq x$. From this it follows that the following weaker disjunction is also true:

$$
\begin{aligned}
& (x \doteq y \wedge \text{gcd}(x, y) \doteq x) \\
\vee \ \ & (x > y \wedge \text{gcd}(x, y) \doteq \text{gcd}(x - y, y)) \\
\vee \ \ & (y > x \wedge \text{gcd}(x, y) \doteq \text{gcd}(x, y - x))
\end{aligned} \tag{11.6}
$$

Note that (11.6) looks remarkably like a functional program for GCD.

12 Completeness for DFOL*

The method of trace sets for proving completeness from Section 10 still applies. Trace sets for DFOL* will have to satisfy the obvious extra conditions. In order to preserve the correspondence between trace sets and open tableau branches, we must adapt the definition of a fair computation rule. A computation rule F for $\mathcal{L}^*_\Sigma$ is fair if it is fair for $\mathcal{L}_\Sigma$, and in addition, the following holds for all branches B in $\boldsymbol{T}_\infty$:

- All formulas of type α^*, β^* occurring on B or in Φ were used to expand B.

We can again prove a trace lemma for DFOL*, in the same manner as before: Again, open branches in the supremum of a fair tableau sequence will correspond to trace sets, and we can satisfy these trace sets in canonical models. The definition of trace sets is extended as follows:

Definition 12.1 *A set Ψ of $\mathcal{L}^*_{\Sigma^*}$ formulas is a* $*$**-trace set** *if the following hold:*

- *Ψ is a trace set,*
- *If $\beta^* \in \Psi$ then at least one $\beta^*_i \in \Psi$.*
- *If $\phi; \psi^*; \chi \in \Psi$, then there is some $n \geq 0$ with $\phi; \psi^m; \chi \notin \Psi$ for all $m > n$. Similarly for $(\!(\phi; \psi^*; \chi)\!)$.*
- *For all ϕ, ψ, χ it holds that $\neg(\phi; \psi^*; \chi) \notin \Psi$.*

Note that the final two requirements are met thanks to our stipulation about closure in the limit. In the same manner as before, we get:

Theorem 12.2 (Completeness for $\mathcal{L}^*$) *For all $\phi, \psi \in \mathcal{L}^*$: if $\phi \models \psi$ then the tableau for $\phi; \neg(\psi)$ closes.*

So we have a complete logic for DFOL*, but of course it comes at a price: we may occasionally get in a refutation loop. However, as our tableau construction examples illustrate, this does hardly affect the usefulness of the calculus.

13 Related Work

Comparison with tableau reasoning for (fragments of) FOL The present calculus for DFOL can be viewed as a more dynamic version of tableau style reasoning for FOL and for modal fragments of FOL. Instead of just checking for valid consequence and constructing counterexamples from open tableau branches, our open tableau branches yield computed answer bindings as an extra. The connection with tableau reasoning for FOL is also evident in the proof method of our completeness theorems. Our calculus can be used for FOL reasoning via the following translation of FOL into DFOL:

$$\begin{array}{rcl}
(P\bar{t})^\bullet & := & P\bar{t} \\
(\neg\phi)^\bullet & := & \neg\phi^\bullet \\
(\phi \wedge \psi)^\bullet & := & \phi^\bullet; \psi^\bullet \\
(\phi \vee \psi)^\bullet & := & \phi^\bullet \cup \psi^\bullet \\
(\exists x\phi)^\bullet & := & (\!(\exists x; \phi^\bullet)\!) \\
(\forall x\phi)^\bullet & := & \neg(\exists x; \neg\phi^\bullet)
\end{array}$$

It is easy to check that for every FOL formula ϕ it holds that $\phi^\bullet = \phi^{\bullet\boxdot}$, i.e., all FOL translations are DFOL tests. Moreover, the translation is adequate in the sense that for every FOL formula ϕ over signature Σ, every Σ-model $\mathcal{M}$, every valuation s for $\mathcal{M}$ it holds that $\mathcal{M} \models_s \phi$ iff ${}_s[\![\phi^\bullet]\!]^{\mathcal{M}}_s$.

Connection with Logic Programming The close connection between tableau reasoning for DFOL and Logic Programming can be seen by developing a DFOL tableau for the following formula set:

$$\forall x A([], x, x), \forall x \forall y \forall z \forall i(A(x, y, z) \rightarrow A([i|x], y, [i|z])), \neg \exists x A([a|[b|[]], [c|[]], x).$$

This will give a tableau for the append relation, with a MGU substitution $\{\boldsymbol{x} \mapsto [a|[b|c|[]]]\}$ that closes the tableau, where $\boldsymbol{x}$ is the universal tableau variable used in the application of the γ rule to $\neg \exists x A([a|[b|[]], [c|[]], x)$. The example may serve as a hint to the unifying perspective on logic programming and imperative programming provided by tableau reasoning for DFOL. In future work, we hope to elaborate the further connections between our delayed substitution rules and constraint logic programming, and between our computational handling of equality and equational reasoning in logic programs.

Comparison with other Calculi for DFOL and for DRT The calculus developed in [12] uses swap rules for moving quantifiers to the front of formulas. The key idea of the present calculus is entirely different: encode dynamic binding in explicit bindings and protect outside environments from dynamic side effects by means of block operations. In a sense, the present calculus offers a full account of the phenomenon of local variable use in DFOL.

Kohlhase [22] gives a tableau calculus for DRT (Discourse Representation Theory, see [21]) that has essentially the same scope as the [12] calculus for DPL: the version of DRT disjunction that is treated is externally static, and the DRT analogue of $\cup$ is not treated.

The Kohlhase calculus follows an old DRT tradition in relying on an implicit translation to standard FOL: see [27] for an earlier example of this. Kohlhase motivates his calculus with the need for (minimal) model generation in dynamic NL semantics. In order to make his calculus generate minimal models, he replaces the rule for existential quantification by a 'scratchpaper' version (well-known from textbook treatments of tableau reasoning; see [20] for further background, and for discussion of non-monotonic consequence based on minimal models generated with this rule): first try out if you can avoid closure with a term already available at the node. If all these attempts result in closure, it does not follow from this that the information at the node is inconsistent, for it may just be that we have 'overburdened' the available terms with demands. So in this case, and only in this case, introduce a new individual.

This 'exhaustion of existing terms' approach has the virtue that it generates 'small' models when they exist, whereas the more general procedure 'always introduce a fresh variable and postpone instantiation' may generate infinite models where finite models exist. Note, however, that the strategy only makes sense for a signature without function symbols, and for a tableau calculus without free tableau variables.

Kohlhase discusses applications in NL processing, where it often makes sense to construct a minimal model for a text, and where the assumption of minimality can be used to facilitate issues of anaphora resolution and presupposition handling.

Comparison with Apt and Bezem's Executable FOL Apt and Bezem present what can be viewed as an exciting new mix of tableau style reasoning and model checking for FOL. Our treatment of equality uses a generalization of a stratagem from their [3]: in the context of a partial variable map θ, they call $v \doteq t$ a θ *assignment* if $v \notin dom(\theta)$, and all variables occurring in t are in $dom(\theta)$. We generalize this on two

counts:

- Because our computation results are bindings (term maps) rather than maps to objects in the domain of some model, we allow computation of non-ground terms as values.
- Because our bindings are total, in our calculus execution of $t_1 \doteq t_2$ atoms never gives rise to an error condition.

It should be noted for the record that the first of these points is addressed in [2]. Apt and Bezem present their work as an underpinning for Alma-0, a language that infuses Modula style imperative programming with features from logic programming (see [4]). In a similar way, the present calculus provides logical underpinnings for Dynamo, a language for programming with an extension of DFOL. For a detailed comparison of Alma-0 and Dynamo we refer the reader to [11].

Connection with WHILE, GCL It is easy to give an explicit binding semantics for WHILE, the favorite toy language of imperative programming from the textbooks (see e.g., [23]), or for GCL, the non-deterministic variation on this proposed by Dijkstra (see, e.g. [9]). DFOL is in fact quite closely related to these, and it is not hard to see that DFOL* has the same expressive power as GCL. Our tableau calculus for DFOL* can therefore be regarded as an execution engine *cum* reasoning engine for WHILE or GCL.

Connection with PDL, QDL There is also a close connection between DFOL* on one hand and propositional dynamic logic (PDL) and quantified dynamic logic (QDL) on the other. QDL is a language proposed in [25] to analyze imperative programming, and PDL is its propositional version. See [28, 24] for complete axiomatizations of PDL, [15] for an exposition of both PDL and QDL, and for a complete (but infinitary) axiomatization of QDL, [19] for an overview, and [18] for a a study of QDL and various extensions. In PDL/QDL, programs are treated as modalities and assertions about programs are formulas in which the programs occur as modal operators. Thus, if A is a program, $\langle A\rangle\phi$ asserts that A has a successful termination ending in a state satisfying ϕ. As is well-known, this cannot be expressed without further ado in Hoare logic.

The main difference between DFOL* and PDL/QDL is that in DFOL* the distinction between formulas and programs is abolished. Everything is a program, and assertions about programs are test programs that are executed along the way, but with their dynamic effects blocked. To express that A has a successful termination ending in a ϕ state, we can just say $(\!(A;\phi)\!)$. To check whether A has a successful termination ending in a ϕ state, try to refute the statement by constructing a tableau for $\neg(A;\phi)$.

To illustrate the connection with QDL and PDL, consider MIX, the first of the two PDL axioms for $*$:

$$[A^*]\phi \to \phi \wedge [A][A^*]\phi. \tag{13.1}$$

Writing this with $\langle A\rangle, \neg, \wedge, \vee$, and replacing $\neg\phi$ by ϕ, we get:

$$\neg(\neg\langle A^*\rangle\phi \wedge (\phi \vee \langle A\rangle\langle A^*\rangle\phi)). \tag{13.2}$$

This has the following DFOL* counterpart:

$$\neg(\neg(A^*;\phi);(\phi \cup (A;A^*;\phi))). \tag{13.3}$$

For a refutation proof of (13.3), we leave out the outermost negation.

$$\neg(A^*;\phi);(\phi\cup(A;A^*;\phi))$$
$$|$$
$$\neg(A^*;\phi)$$
$$(\phi\cup(A;A^*;\phi))$$
$$|$$
$$\neg\phi$$
$$\neg(A;A^*;\phi)$$
$$\phi \qquad\qquad (A;A^*;\phi)$$
$$\times \qquad\qquad \times$$

The tableau closes, so we have proved that (13.3) is a DFOL* theorem (and thus, a DFOL* validity).

We will also derive the validity of the DFOL* counterpart to IND, the other PDL axiom for $*$:

$$(\phi\wedge[A^*](\phi\rightarrow[A]\phi))\rightarrow[A^*]\phi. \tag{13.4}$$

Equivalently, this can be written with only $\langle A\rangle,\neg,\wedge,\vee$, as follows:

$$\neg(\phi\wedge\neg\langle A^*\rangle(\phi\wedge\langle A\rangle\neg\phi)\wedge\langle A^*\rangle\neg\phi). \tag{13.5}$$

The DFOL* counterpart of (13.5) is:

$$\neg(\phi;\neg(A^*;\phi;A;\neg\phi);A^*;\neg\phi). \tag{13.6}$$

We will give a refutation proof of (13.6) in two stages. First, we show that (13.7) can be refuted for any $n\geq 0$, and next, we use this for the proof of (13.6).

$$\phi;\neg(A^*;\phi;A;\neg\phi);A^n;\neg\phi. \tag{13.7}$$

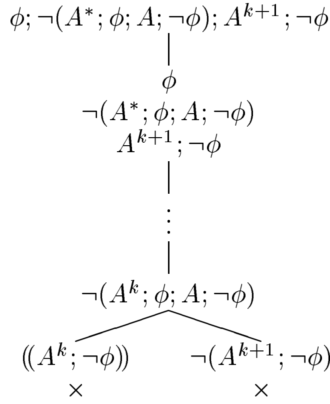
Here is the case of (13.7) with $n=0$:

$$\phi;\neg(A^*;\phi;A;\neg\phi);\neg\phi$$
$$|$$
$$\phi$$
$$\neg(A^*;\phi;A;\neg\phi)$$
$$\neg\phi$$
$$\times$$

Bearing in mind that A is a dynamic action and ϕ is a test, we can apply the rule of Negation Splitting to formulas of the form $\neg(A^n;\phi;A;\neg\phi)$, as follows:

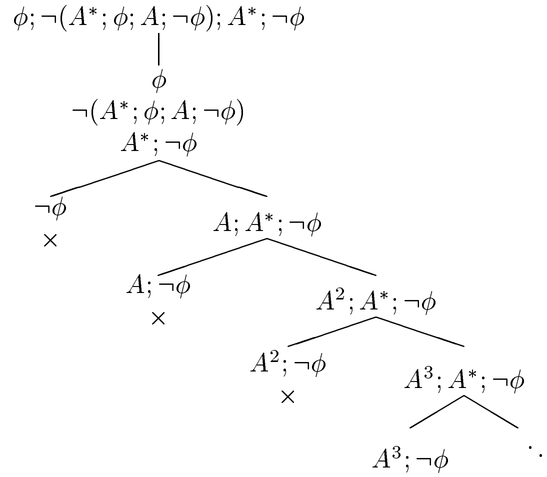
$$\neg(A^n;\phi;A;\neg\phi)$$
$$((A^n;\neg\phi)) \qquad \neg(A^{n+1};\neg\phi)$$

Note that $\neg(A^n;\phi;A;\neg\phi)$ can be derived from $\neg(A^*;\phi;A;\neg\phi)$ by n applications of the α^* rule. Using this, we get the following refutation tableau for the case of (13.7) with $n=k+1$:



The left-hand branch closes because of the refutation of $\phi; \neg(A^*; \phi; A; \neg\phi); A^k; \neg\phi$, which is given by the induction hypothesis.

Next, use these refutations of $\neg\phi$, $A; \neg\phi$, $A^2; \neg\phi$, ..., to prove (13.6) by means of a refutation in the limit, as follows:

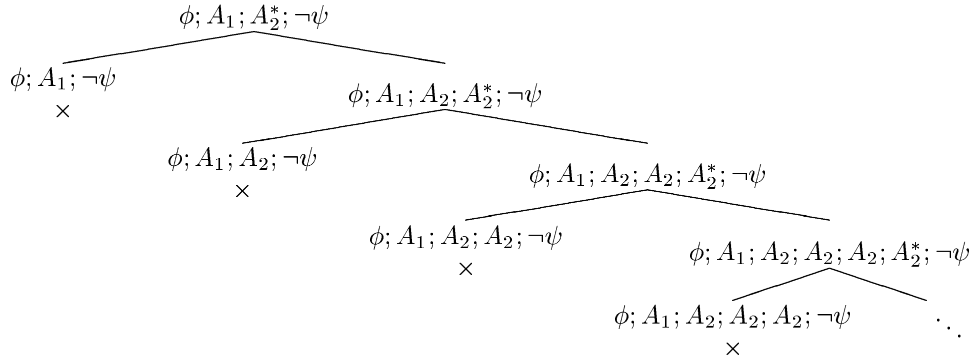


This closed tableau establishes (13.6) as a DFOL* theorem. That closure in the limit is needed to establish the DFOL* induction principle is not surprising. The DFOL * rules express that * computes a fix-point, while the fact that this fix-point is a *least* fix-point is captured by the stipulation about closure in the limit. The induction principle (13.6) hinges on the fact that * computes a least fix-point.

Goldblatt [14, 15] develops an infinitary proof system for QDL with the following key rule of inference:

$$\text{If } \phi \rightarrow [A_1; A_2^n]\psi \text{ is a theorem for every } n \in \mathbb{N}, \text{ then } \phi \rightarrow [A_1; A_2^*]\psi \text{ is a theorem.} \tag{13.8}$$

To see how this is related to the present calculus, assume that one attempts to refute $\phi \rightarrow [A_1; A_2^*]\psi$, or rather, its DFOL* counterpart $\neg(\phi; A_1; A_2^*; \neg\psi)$, on the assumption that for any $n \in \mathbb{N}$ there exists a refutation of $\phi; A_1; A_2^n; \neg\psi$.



We can close off the $\phi; A_1; A_2^n; \neg\psi$ branches by the assumption that there exist refutations for these, for every $n \in \mathbb{N}$. The whole tableau gives an infinite β^* development, and the infinite branch closes in the limit, so the tableau closes, thus establishing that in the DFOL* calculus validity of $\neg(\phi; A_1; A_2^*; \neg\psi)$ follows from the fact that $\neg(\phi; A_1; A_2^n; \neg\psi)$ is valid for every $n \in \mathbb{N}$.

14 Conclusion

Starting out from an analysis of binding in dynamic FOL, we have given a tableau calculus for reasoning with DFOL. The format for the calculus and the role of explicit bindings for computing answers to queries were motivated by our search for logical underpinnings for programming with (extensions of) DFOL. The DFOL tableau calculus presented here constitutes the theoretical basis for *Dynamo*, a toy programming language based on DFOL. The versions of *Dynamo* implemented so far implement tableau reasoning for DFOL with respect to a fixed model: see [11].

To find the answer to a query, given a formula ϕ considered as *Dynamo* program data, *Dynamo* essentially puts the tableau calculus to work on a formula ϕ, all the while checking predicates with respect to the fixed model of the natural numbers, and storing values for variables from the inspection of equality statements. If the tableau closes, this means that ϕ is inconsistent (with the information obtained from testing on the natural numbers), and *Dynamo* reports 'false'. If the tableau remains open, *Dynamo* reports that ϕ is consistent (again with the information obtained from inspecting predicates on the natural numbers), and lists the computed bindings for the output variables at the end of the open branches. But the *Dynamo* engine also works for general tableau reasoning, and for general queries. The literals collected along the open branches together with the explicit bindings at the trail ends constitute the computed answers.

Dynamo can be viewed as a combined engine for program execution and reasoning. We are currently working on an new implementation of *Dynamo* that takes the insights reported above into account. The advantages of the combination of execution and reasoning embodied in *Dynamo* should be evident from our examples of strongest postcondition generation in Section 9. To our knowledge, this use of dynamic first order logic for analyzing imperative programming by means of calculating trace sets is new. We claim that our calculus opens the road to a more intuitive way of reasoning about imperative programs, and we hope to develop automated reasoning tools for

program analysis based on it.

Finally, since natural language semantics is a key application area of dynamic variations on first order logic, we expect that both the calculus itself and its implementation in the form of an improved execution mechanism for *Dynamo* also have a role to play in a truly computational semantics for natural language.

Acknowledgments

The research for this paper was sponsored by *Spinoza Logic In Action*. Thanks to Johan van Benthem, Balder ten Cate, Anne Kaldewaij, Fairouz Kamareddine, Michael Kohlhase, Maarten Marx, Joachim Niehren, Kees Vermeulen, Albert Visser and Joe Wells for stimulating discussion and helpful criticism. Two anonymous reviewers of this journal made suggestions that prompted a complete overhaul of the presentation. Proposition (3.4) was triggered by a question from Krzysztof Apt.

References

[1] K.R. Apt. *From Logic Programming to Prolog.* Prentice Hall, 1997.

[2] K.R. Apt. A denotational semantics for first-order logic. In *Proc. of the Computational Logic Conference (CL2000)*, Notes in Artificial Intelligence 1861, pages 53–69. Springer, 2000.

[3] K.R. Apt and M. Bezem. Formulas as programs. In K.R. Apt, V. Marek, M. Truszczyski, and D.S. Warren, editors, *The Logic Programming Paradigm: a 25 Years Perspective*, pages 75–107. Springer Verlag, 1999. Paper available as `http://xxx.lanl.gov/abs/cs.LO/9811017`.

[4] K.R. Apt, J. Brunekreef, V. Partington, and A. Schaerf. Alma-0: An imperative language that supports declarative programming. *ACM Toplas*, 20:1014–1066, 1998.

[5] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

[6] J. van Benthem. *Exploring Logical Dynamics.* CSLI & Folli, 1996.

[7] J. van Benthem and J. van Eijck. The dynamics of interpretation. *Journal of Semantics*, 1(1):3–20, 1982.

[8] M. D'Agostino, D.M. Gabbay, R. Hähnle, and J. Posegga, editors. *Handbook of Tableau Methods.* Kluwer, Dordrecht, 1999.

[9] E.W Dijkstra and C.S. Scholten. *Predicate Calculus and Program Semantics.* Texts and Monographs in Computer Science. Springer-Verlag, 1990.

[10] H.C. Doets. *From Logic to Logic Programming.* MIT Press, Cambridge, Massachusetts, 1994.

[11] J. van Eijck. Programming with dynamic predicate logic. Technical Report CT-1998-06, ILLC, 1998. Available from `www.cwi.nl/~jve/dynamo`.

[12] J. van Eijck. Axiomatising dynamic logics for anaphora. *Journal of Language and Computation*, 1:103–126, 1999.

[13] M. Fitting. *First-order Logic and Automated Theorem Proving; Second Edition.* Springer Verlag, Berlin, 1996.

[14] R. Goldblatt. *Axiomatising the Logic of Computer Programming.* Springer, 1982.

[15] R. Goldblatt. *Logics of Time and Computation, Second Edition, Revised and Expanded*, volume 7 of *CSLI Lecture Notes.* CSLI, Stanford, 1992 (first edition 1987). Distributed by University of Chicago Press.

[16] J. Groenendijk and M. Stokhof. Dynamic predicate logic. *Linguistics and Philosophy*, 14:39–100, 1991.

[17] R. Hähnle. Tableaux and related methods. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning.* Elsevier Science Publishers, to appear, 2001.

[18] D. Harel. *First-Order Dynamic Logic.* Number 68 in Lecture Notes in Computer Science. Springer, 1979.

[19] D. Harel. Dynamic logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, pages 497–604. Reidel, Dordrecht, 1984. Volume II.

[20] J. Hintikka. Model minimization — an alternative to circumscription. *Journal of Automated Reasoning*, 4:1–13, 1988.

[21] H. Kamp. A theory of truth and semantic representation. In J. Groenendijk et al., editors, *Formal Methods in the Study of Language*. Mathematisch Centrum, Amsterdam, 1981.

[22] M. Kohlhase. Model generation for Discoure Representation Theory. In *ECAI Proceedings*, 2000. Available from `http://www.ags.uni-sb.de/~kohlhase/`.

[23] H.R. Nielson and F. Nielson. *Semantics with Applications*. John Wiley and Sons, 1992.

[24] R. Parikh. The completeness of propositional dynamic logic. In *Mathematical Foundations of Computer Science 1978*, pages 403–415. Springer, 1978.

[25] V. Pratt. Semantical considerations on Floyd–Hoare logic. *Proceedings 17th IEEE Symposium on Foundations of Computer Science*, pages 109–121, 1976.

[26] W. Reif. The KIV-approach to software verification. In M. Broy and S. Jähnichen, editors, *KORSO: Methods, Languages, and Tools for the Construction of Correct Software*, Springer LNCS 1009, pages 339–368, 1995.

[27] C. Sedogbo and M. Eytan. A tableau calculus for DRT. *Logique et Analyse*, 31:379–402, 1988.

[28] K. Segerberg. A completeness theorem in the modal logic of programs. In T. Traczyck, editor, *Universal Algebra and Applications*, pages 36–46. Polish Science Publications, 1982.

[29] R. Smullyan. *First-order logic*. Springer, Berlin, 1968.

[30] Y. Venema. A modal logic of quantification and substitution. In L. Czirmaz, D.M. Gabbay, and M. de Rijke, editors, *Logic Colloquium '92*, Studies in Logic, Language and Computation, pages 293–309. CSLI and FOLLI, 1995.

[31] A. Visser. Contexts in dynamic predicate logic. *Journal of Logic, Language and Information*, 7(1):21–52, 1998.

[32] A. Visser. A note on substitution in dynamic semantics. Unpublished draft, Utrecht University, 2000.

Received September 8, 2000. Revised: December 9, 2000, January 30, 2001

Theorem Proving in Infinitesimal Geometry

JACQUES D. FLEURIOT, *Division of Informatics, University of Edinburgh, 80 South Bridge, Edinburgh EH1 1HN, UK.*
Email: jdf@dai.ed.ac.uk

Abstract

This paper describes some of the work done in our formal investigation of concepts and properties that arise when infinitely small and infinite notions are introduced in a geometry theory. An algebraic geometry theory is developed in the theorem prover Isabelle using real and hyperreal vectors. We use this to investigate some new geometric relations as well as ways of rigorously mechanizing geometric proofs that involve infinitesimal and infinite arguments. We follow a strictly definitional approach and build our theory of vectors within the nonstandard analysis framework developed in Isabelle.

Keywords: nonstandard analysis, geometry, hyperreals, theorem proving, higher order logic, Isabelle

1 Introduction

In our previous work on the mechanization of Newton's *Principia*, we introduced, through a combination of techniques from geometry theorem proving (GTP) and nonstandard analysis (NSA), the notion of an infinitesimal geometry in which quantities can be infinitely small [11, 12]. The main aim was to capture and mechanize the limit or *ultimate* notions used by Newton in his proofs, while respecting as much as possible his original geometric arguments.

Our formalization task, within the interactive framework of Isabelle, was made possible through the use of concepts from powerful— yet geometrically intuitive— GTP techniques known as the signed area and full-angle methods [3, 4]. These methods were highly adequate to our goals as they provided us with lemmas powerful enough to prove the results we wanted but also used geometric notions such as areas and ratios of segments that were directly relevant to Newton's proofs.

In the current work, however, we depart to some extent from the framework already established in Isabelle for geometry. Our aim, now, is to *formally* explore the properties of the infinitesimal geometry theory developed in Isabelle. To this end, we formulate an alternative treatment of geometry based on the notions of hyperreal vectors. We want to provide a rigorous yet powerful theory that can capture formally the properties of our geometry, as well as provide a secure foundations for our previous work [11, 12, 10].

Moreover, the approach we describe in this paper also differs from that previously adopted in that it is fully definitional. In other words, we now formally define and derive *all* mathematical notions rather than postulate any of them. This approach guarantees consistency, which cannot be ensured when axioms are introduced (see Section 3.3 for a brief overview of this methodology).

As a further motivation for the current work, our rigorous development within

 ©Oxford University Press

Isabelle can be viewed as providing formal justifications for the basic rules and lemmas used in the automatic GTP methods of Chou et al. [3, 4, 2]. Indeed, as outlined later in this paper, we formally derive in our definitional theory many of the rules used in their geometry theorem provers. In addition, since the nonstandard approach presented in this work can be used to prove standard geometry theorems, it should be of interest to the mechanical geometry theorem proving community. Our approach might provide ways of extending current automatic methods to produce proofs that incorporate infinitely small notions without resulting in degeneracy.

In what follows, we first present some (historical) motivation for the existence of a geometry involving infinitesimal notions (Section 2). We then give a brief overview of Isabelle in which this work is carried out (Section 3). We also introduce some of the basic (mechanized) notions from nonstandard analysis that will prove useful to our discussion(Section 4). In particular, we look briefly at the construction of the hyperreal numbers and how operations like addition are defined and also at the nonstandard extension of functions. Next, we give an overview of the vector theory developed in Isabelle (Section 5) by reviewing the vector algebra, the vectorial definitions used for familiar geometric properties, and some of the infinitesimal geometry theorems that follow. We then describe some of the novel infinitesimal geometric concepts formalized in the work so far (Section 6). We then describe a new approach, based on nonstandard methods, that can be used for proving standard geometry theorems (Section 7). Finally, we outline some of the further work currently in the pipeline (Section 8) and share some of the conclusions we have reached so far (Section 9).

In the next section, we present some motivation for our geometry by briefly examining the related notion of non-Archimedean geometry.

2 Non-Archimedean Geometry

The *Axiom of Archimedes* or *Axiom of Continuity* from Hilbert's *Foundations of Geometry* [16] may be stated as follows:

> Let A, B, C, and D be four distinct points. Then on the ray AB there is a finite set of distinct points, A_1, A_2,..., A_n such that each segment A_iA_{i+1} is congruent to the segment CD and such that B is between A and A_n.

This means that given any line segment of length l and any measure m, there exists an integer n such that n units of measure yield a line segment greater than the given line segment i.e. $l < n \cdot m$. Geometrically speaking, this means that the length of a line has no limit, which is a tacit assumption of Euclid. This axiom of Hilbert can therefore be viewed as stating that the points on the line are in one-to-one correspondence with the real numbers $\mathbb{R}$.

After introducing the various groups of axioms, Hilbert proceeds to show their consistency and mutual independence. This is done by interpreting every geometric concept arithmetically and making sure that all the axioms are satisfied in the interpretation. For example, a point is identified with the ordered pair of real numbers (a,b) and a line with the ratio $(u{:}v{:}w)$ in which u and v are both non-zero. A point lies on a line if $ua + vb + w = 0$. Properties such as convergence are interpreted algebraically by means of the expressions for translation and rotation of analytic geometry. Thus, a model is constructed for the axioms of geometry and any contradiction deduced

from these would mean that the axioms of arithmetic are inconsistent.

The possibility of a non-Archimedean geometry is exposed when proving the mutual independence of Hilbert's sets of axioms. Indeed, it is possible to construct a model that satisfies all the various axioms except the Axiom of Archimedes. In such a geometry, our measure m can be laid off successively upon our line segment of length l an arbitrary number of times without ever reaching the end point of the line. This geometry might be seen, intuitively, as one in which infinitesimal notions are allowed. Of course, the most famous example of an axiom being denied in geometry is that of the parallel axiom, which leads to non-Euclidean geometry.

It is worth noting that one of the first to attempt a systematic investigation of non-Archimedean geometry was the Italian mathematician Veronese in his *Fundamenti di Geometria*. As observed by Fisher [9], his work was often unacknowledged by contemporary mathematicians such as Hilbert and Poincaré and only recently have historians given its influence due recognition. Veronese's poor and tortuous exposition has been blamed to some extent for this.

In his review of Hilbert's Foundations of Geometry, Henri Poincaré makes the following important observation about non-Archimedean geometry [21]:

> ... the coordinates of a point would be measured not by ordinary numbers but by non-Archimedean numbers, while the usual operations of the straight lines and the plane would hold, as well as the analytic expressions for angles and lengths. It is clear that in this space all the axioms would remain true except that of Archimedes.

And moreover, he notes

> On every straight line new points would be interpolated between ordinary points.

This matches our approach in which we effectively replace the real number line with a hyperreal one. The hyperreal numbers of NSA (which we review in Section 4) thus correspond to Poincaré's non-Archimedean numbers. Poincaré also gives a geometric example where an ordinary line is compared with a non-Archimedean one:

> If, for example, D_0 is an ordinary straight line, and D_1 the corresponding non-Archimedean straight line; if P is any ordinary point of D_0, and if this point divides D_0 into two half rays S and S' (I add, for precision, that I consider P as not belonging to either S or S') then there will be on D_1 an infinity of new points as well between P and S as between P and S'. Then there will be on D_1 an infinity of new points which will lie to the right of all the ordinary points of D_0. In short, our ordinary space is only a part of the non-Archimedean space.

This geometrical representation means that points can be infinitely close to each other on line D_1. Indeed, the first infinity of new points mentioned by Poincaré corresponds to those infinitely close to P. And then, we also have the new points on D_1 that lie beyond those of D_0. These points to the right of all of the points of D_0 thus correspond to the infinite hyperreals. These observations motivate the establishment of a one-to-one correspondence f between the hyperreals and a line L instead of the usual correspondence with the reals. A *coordinate system*, f, for L is then such that each point P on it has a unique hyperreal coordinate given by $x = f(P)$.

We next give a brief introduction to Isabelle and to the HOL object logic in which this work was carried out.

3 Isabelle/HOL

Isabelle [19] is a generic theorem prover, written in ML, into which the user can encode their own object-level logics. Examples of such object logics are higher order logic (HOL), Zermelo-Fraenkel set theory (ZF), and first order logic (FOL). Terms from the object logics are represented and manipulated in Isabelle's intuitionistic higher order meta-logic, which supports polymorphic typing.

3.1 Theories in Isabelle

Isabelle's theories provide a hierarchical organization for the syntax, declarations and axioms of a mathematical development and can be developed using theory definition files [19]. A typical theory file will organize the definitions of types and functions. It may also contain the primitive axioms that are asserted (without proofs) by the user. A particular theory will usually collect (in a separate file) the proven named theorems and make them available to all its children theories.

The meta-level connectives are implication ($\Longrightarrow$), universal quantifier and equality. Throughout the presentation, we will be using mostly conventional mathematical notations when describing our development. However, there are cases where we might use the ASCII notations actually used to express terms and rules in Isabelle as explicit examples.

An inference rule with n premises or antecedents has the following form in Isabelle:

$$[|\phi_1; \ldots ; \phi_n|] \Longrightarrow \psi$$

This abbreviates the nested implication $\phi_1 \Longrightarrow (\ldots \phi_n \Longrightarrow \psi \ldots)$. Such a rule can also be viewed as the proof state with subgoals $\phi_1, \ldots, \phi_n$ and main goal ψ [19]. Alternatively, this can be viewed as meaning "if $\phi_1 \wedge \cdots \wedge \phi_n$ then ψ".

3.2 Higher Order Logic in Isabelle

One of Isabelle's logics is HOL, a higher order logic that supports polymorphism and type constructors. Isabelle/HOL is based on Gordon's HOL theorem prover [15] which itself originates from Church's paper [5]. Isabelle/HOL is well developed and widely used. It has a wide library of theories defined in it including the naturals, integers and real numbers, set theory, well-founded recursion, inductive definitions, and equivalence relations.

Though Isabelle is mainly used interactively as a proof assistant, it also provides substantial support for automation. It has a generic simplification package, which is set up for many of the logics including HOL. Isabelle's simplifier performs conditional and unconditional rewritings and makes use of context information [19]. The user is free to add new rules to the simplification set (the *simpset*) either permanently or temporarily. Isabelle also provides a number of generic automatic tactics that can execute proof procedures in the various logics. The automatic tactics provided by Isabelle's *classical reasoner* include a fast tableau prover called `Blast_tac` coded

directly in ML and `Auto_tac` which attempts to prove all subgoals by a combination of simplification and classical reasoning. Other powerful theorem proving tactics include those which, unlike `Blast_tac`, construct proofs directly in Isabelle: for example, `Fast_tac` implements a depth-first search automatic tactic. In addition to these various tools, Isabelle/HOL now also provides decision procedures for linear arithmetic that greatly simplify many proofs over the real numbers.

3.3 The HOL Methodology

The HOL methodology, which derives from work done by Gordon in the HOL theorem prover [15], admits only conservative extensions to a theory. This means, as we already mentioned in the introduction, defining and deriving the required mathematical notions rather than postulating them. The definitional approach of HOL requires that assertions are proved about some model instead of being postulated. Such a rigorous definitional extension guarantees consistency, which cannot be ensured when axioms are introduced. With regards to the foundations of infinitesimals and of our geometry, the definitional approach is certainly advisable when one considers the numerous inconsistent axiomatizations that have been proposed in the past [6].

4 A Few Concepts from Nonstandard Analysis

An immediate consequence of our decision to formalize nonstandard rather than standard analysis is the extra amount of work spent on number constructions. The ultrapower construction of the hyperreals, for example, first required proving Zorn's Lemma and developing a theory of filters and ultrafilters for Isabelle/HOL. We have described details of the construction elsewhere [10, 13], and so will only outline a few of the aspects relevant to this paper in what follows.

4.1 On the Construction

The construction of the hyperreals (denoted by $\mathbb{R}^*$) resembles to some extent that of the reals from the rationals using equivalence classes induced by Cauchy sequences. In this case, however, a free ultrafilter $U_{\mathbb{N}}$ over the natural numbers is used to partition the set of all sequences of real numbers into equivalence classes. The free ultrafilter $U_{\mathbb{N}}$, whose existence is proved using Zorn's Lemma, is a collection of subsets of $\mathbb{N}$ with the following properties (amongst others)[13, 10]:

$$\emptyset \notin U_{\mathbb{N}} \text{ and } \mathbb{N} \in U_{\mathbb{N}} \qquad X \in U_{\mathbb{N}} \Longrightarrow \neg\, \mathtt{finite}\, X$$
$$X \in U_{\mathbb{N}} \wedge Y \in U_{\mathbb{N}} \Longrightarrow X \cap Y \in U_{\mathbb{N}} \qquad X \in U_{\mathbb{N}} \iff -X \notin U_{\mathbb{N}}$$
$$X \in U_{\mathbb{N}} \wedge X \subseteq Y \Longrightarrow Y \in U_{\mathbb{N}}$$

In Isabelle, the following equivalence relation on sequences of real numbers is then defined:[1]

$$\mathtt{hyprel} :: ((\mathtt{nat} \Rightarrow \mathtt{real}) * (\mathtt{nat} \Rightarrow \mathtt{real}))\ \mathtt{set}$$
$$\mathtt{hyprel} \equiv \{p.\ \exists rs.\ p = (r, s) \wedge \{n.\ r(n) = s(n)\} \in U_{\mathbb{N}}\}$$

[1] The Isabelle notation $a{::}\tau$ denotes that a is of type τ

The set of equivalence classes, that is the quotient set, arising from `hyprel` is used to define the new type `hypreal` denoting the hyperreals:

$$\texttt{hypreal} \equiv \{x{::}(\texttt{nat} \Rightarrow \texttt{real}).\texttt{True}\}/\texttt{hyprel}$$

Thus, it follows from the definition of `hyprel` that for two hyperreals to be equal, the corresponding entries in their equivalence class representatives must be equal at an infinite number of positions. This is because $U_{\mathbb{N}}$ cannot contain any finite set. Once the new type has been introduced, Isabelle provides coercion functions — the abstraction and representation functions — that enable the basic operations to be defined. In this particular case, the functions

$$\begin{array}{l}\texttt{Abs_hypreal} :: (\texttt{nat} \Rightarrow \texttt{real})\,\texttt{set} \Rightarrow \texttt{hypreal}\\ \texttt{Rep_hypreal} :: \texttt{hypreal} \Rightarrow (\texttt{nat} \Rightarrow \texttt{real})\,\texttt{set}\end{array}$$

are added to the theory such that `hypreal` and $\{x{::}\ \texttt{nat} \Rightarrow \texttt{real}.\ \texttt{True}\}/\texttt{hyprel}$ are isomorphic by `Rep_hypreal` and its inverse `Abs_hypreal`.

The familiar operations (addition, subtraction, multiplication, inverse) and the ordering relation on the new type `hypreal` are then defined in terms of pointwise operations on the underlying sequences. For example, let $[\langle X_n \rangle]$ denote the equivalence class (i.e. hyperreal) containing $\langle X_n \rangle$ then multiplication is defined by

$$[\langle X_n \rangle] \cdot [\langle Y_n \rangle] \equiv [\langle X_n \cdot Y_n \rangle] \tag{4.1}$$

or, more specifically, in Isabelle as:

$$\begin{array}{rl} P\ \cdot\ Q \equiv & \texttt{Abs_hypreal}\ (\bigcup X \in \texttt{Rep_hypreal}(P).\\ & \bigcup Y \in \texttt{Rep_hypreal}(Q).\,\texttt{hyprel}\texttt{``}\{\lambda n.\, X\,n\ \cdot\ Y\,n\})\end{array}$$

where

$$\bigcup x \in A.\, B[x] \equiv \{y.\, \exists x \in A.\, y \in B\}\ \text{(union of family of sets).}$$
$$r\texttt{``}s \equiv \{y.\, \exists x \in s.\, (x,y) \in r\}\ \text{(image of set } s \text{ under relation } r\text{).}$$

Equation (4.1) above is in fact proved as a theorem. All the expected field properties of the hyperreals are easily established since they follow nicely from the corresponding properties of the reals. We define an embedding of the reals in the hyperreals by having the following map in Isabelle:

$$\begin{array}{l}\texttt{hypreal_of_real} :: \texttt{real} \Rightarrow \texttt{hypreal}\\ \texttt{hypreal_of_real}\ r \equiv \texttt{Abs_hypreal}\ (\texttt{hyprel}\texttt{``}\{\lambda n{::}\texttt{nat}.\, r\})\end{array}$$

In other words, we represent each real number r in $\mathbb{R}^*$ by the equivalence class $[\langle r, r, r, \ldots \rangle]$. The properties of the embedding function, with respect to multiplication, addition and so on, follow trivially since they are just special cases of the operations on the hyperreals. In what follows, we will denote an embedded real r by $\underline{r}$ unless we use the Isabelle embedding function explicitly.

The ordering relation on the hyperreals, for its part, is defined as follows:

$$\begin{array}{rl} P\ <\ Q \equiv & \exists X \in \texttt{Rep_hypreal}\ P.\\ & \exists Y \in \texttt{Rep_hypreal}\ Q.\, \{n.\, X\,n < Y\,n\} \in U_{\mathbb{N}}\end{array}$$

We prove the corresponding simplification theorem expressing the order relation in terms of equivalence classes of sequences of real numbers:

$$\texttt{Abs_hypreal}\,(\texttt{hyprel}\text{\^{}\^{}}\{X\,n\}) < \texttt{Abs_hypreal}\,(\texttt{hyprel}\text{\^{}\^{}}\{Y\,n\}) \iff \{n.\,X\,n < Y\,n\} \in U_{\mathbb{N}}$$

With this done, it is straightforward to show that $<$ is total [10, 13]. This means that $\mathbb{R}^*$ is a total ordered field.

This section has provided a brief summary of the construction of the nonstandard numbers. Our main intention was to illustrate some of the key concepts of our definitional approach. However, this overview will also be useful to our subsequent exposition as the construction of hyperreal vectors is almost identical to that of the hyperreals: one simply considers equivalence classes of sequences of real vectors rather than sequences of real numbers (see Section 5).

4.2 Nonstandard Numbers

The embedding function enables us to define the set of embedded reals `SReal` explicitly, and prove that it is a proper subfield of $\mathbb{R}^*$. The proof shows that the well-defined hyperreal $[\langle 1, 2, 3, \ldots\rangle]$ (denoted by ω) cannot be equal to any of the embedded reals as no singleton set is allowed in $U_{\mathbb{N}}$. Once the embedding is defined and various of its properties proved, we formalize the definitions characterizing the various types of numbers that make up the new extended field:

$$\begin{array}{l}\texttt{Infinitesimal} \equiv \{x.\ \forall r \in \texttt{SReal}.\ 0 < r \to \texttt{abs}\ x < r\} \\ \texttt{Finite} \equiv \{x.\ \exists r \in \texttt{SReal}.\ \texttt{abs}\ x < r\} \\ \texttt{Infinite} \equiv -\texttt{Finite}\end{array}$$

With this done, a number of theorems are proved, including:

$$\frac{x \in \mathsf{Infinitesimal} \quad y \in \mathsf{Infinitesimal}}{x \ \mathsf{op}\ y \in \mathsf{Infinitesimal}} \qquad \frac{x \in \mathsf{Finite} \quad y \in \mathsf{Finite}}{x \ \mathsf{op}\ y \in \mathsf{Finite}}$$

where op is $+$, $-$, or $\cdot$ (i.e. both sets are subrings of $\mathbb{R}^*$). Other Isabelle theorems proved include amongst many others:

$$\frac{x \in \mathsf{Infinitesimal} \quad y \in \mathsf{Finite}}{x \cdot y \in \mathsf{Infinitesimal}} \qquad \frac{z \in \mathsf{Infinitesimal} \quad \underline{x} < \underline{y}}{\underline{x} + z < \underline{y}}$$

A substantial number of theorems are proved about the properties of the hyperreals and their inter-relationships. In addition, we use our free ultrafilter to extend the natural numbers and construct the *hypernatural* numbers, $\mathbb{N}^*$. This additional type of nonstandard numbers provides us with infinitely large numbers greater than all the members of $\mathbb{N}$. The set of infinite hypernaturals is denoted by `HNatInfinite` in Isabelle. We also define the function `hypnat_of_nat`, an embedding of the natural numbers into the hypernaturals [10].

4.3 Infinitely Close Relation and Standard Part Theorem

In addition to the nonstandard numbers, we need to mechanize a few more important concepts for us to have with an adequate framework for our proofs. Firstly, we define

the crucial *infinitely close* relation $\approx$:

$$x \approx y \equiv x - y \in \texttt{Infinitesimal}$$

This is an equivalence relation about which we prove a number of properties such as:[2]

$$[|a \approx b; c \approx d|] \Longrightarrow a + c \approx b + d$$
$$[|s \in \texttt{SReal}; b \in \texttt{SReal}|] \Longrightarrow (a \approx b) = (a = b) \quad (4.2)$$
$$s \in \texttt{Finite} \Longrightarrow \exists! r.\ r \in \texttt{SReal} \wedge s \approx r \quad (4.3)$$
$$[|a \approx b; c \in \texttt{Finite}|] \Longrightarrow a \cdot c \approx b \cdot c \quad (4.4)$$

Theorem (4.3)above is known as the *Standard Part Theorem* and is especially important as it enables us to formalize the notion of *standard part*. The standard part of a finite nonstandard number is defined as the unique real number infinitely close to it. The actual definition in Isabelle uses the Hilbert choice operator ϵ and returns a number of type `real` rather than an embedded real:

$$\texttt{str} :: \texttt{hypreal} \Rightarrow \texttt{real}$$
$$\texttt{str}\ x \equiv (\epsilon r. x \in \texttt{Finite} \wedge \texttt{hypreal_of_real}\ r \approx x)$$

All the important properties of the standard part operator are proved. These include, for example:

$$\frac{}{\mathsf{str}\ \underline{x} = x} \qquad \frac{x \in \texttt{Finite}}{\mathsf{str}\ \underline{x} \approx x} \qquad \frac{x \in \texttt{Finite} \quad y \in \texttt{Finite}}{(x \approx y) = (\mathsf{str}\ x = \mathsf{str}\ y)}$$

In Section 5.3, we present an extension of the infinitely close relation to hyperreal vectors and use it to investigate the various notions formalized by this work.

4.4 Nonstandard Extensions

Nonstandard extensions provide systematic ways through which sets and functions defined on the reals are extended to the hyperreals (a process sometimes known as the $*$-transform [17]).

In particular, if f is a function from $\mathbb{R}$ to $\mathbb{R}$, then it can be extended to a function f^* from $\mathbb{R}^*$ to $\mathbb{R}^*$ by the following rule: $x = [\langle X_n \rangle] \in \mathbb{R}^*$ maps into $y = [\langle Y_n \rangle] = f^*(x) \in \mathbb{R}^*$ if and only if $\{n \in \mathbb{N}.\ f(X_n) = Y_n\} = U_{\mathbb{N}}$. In Isabelle, this is rendered as:

$$\texttt{*f*} :: (\texttt{real} \Rightarrow \texttt{real}) \Rightarrow \texttt{hypreal} \Rightarrow \texttt{hypreal}$$
$$\texttt{*f*}\ f\ x \equiv \texttt{Abs_hypreal}\ (\bigcup X \in \texttt{Rep_hypreal}(x).\ \texttt{hyprel\^{}\^{}}\{\lambda n.\ f(X n)\})$$

Thus, the nonstandard extension operator provides a generic way through which, given a function taking standard arguments, we can define an analogous one that accepts nonstandard arguments. In what follows, we will denote the nonstandard

[2] $\exists! x.\ P$ stands for the unique existence quantifier, and the "if and only if"connective is denoted by = in Isabelle/HOL.

extension of a given real function f either by f^* or by its equivalent Isabelle notation $(*\texttt{f}*f)$. We prove this important simplification theorem:

$$(*\texttt{f}*\ f)\quad (\texttt{Abs_hypreal}\,(\texttt{hyprel}\texttt{``}\{\lambda n.\ Xn\})) = \\ (\texttt{Abs_hypreal}\,(\texttt{hyprel}\texttt{``}\{\lambda n.\ f(Xn)\}))$$

In other words, we have that $f^*[\langle X_n\rangle] = [\langle f(X_n)\rangle]$. This is useful as it allows us to formalize definitions and prove properties of nonstandard functions by couching them in terms of the corresponding real functions and our free ultrafilter. We easily prove a number of theorems about nonstandard extensions such as $f^*(\underline{r}) = \underline{f(r)}$ and $f^*(x) + g^*(x) = (\lambda u.\ f(u) + g(u))^*(x)$. We will come across others as we further outline our formalization of analysis.

We also extend functions from $\mathbb{N}$ to $\mathbb{R}$: given such a function s, its $*$-transform is the function $s^* : \mathbb{N}^* \rightarrow \mathbb{R}^*$ where $s^*([\langle X_n\rangle]) = [\langle s(X_n)\rangle]$ for any $[\langle X_n\rangle] \in \mathbb{N}^*$. In Isabelle, the nonstandard extension is denoted by $(*\texttt{fNat}*\ s)$ and is useful in the formalization of sequences, for example [10, 13].

We now have enough the basic notions to describe the hyperreal vector and the infinitesimal geometric theories.

5 A Mechanized Theory of Hyperreal Vectors

Apart from using an interactive (hence slower) approach to GTP, the current work also differs from the traditional automated approach by residing within the higher-order logic framework of Isabelle/HOL [20]. One of the main reasons for choosing Isabelle/HOL is that it provides a rigorous framework for the formalization of the infinitesimal— a notoriously difficult task. The suitability of Isabelle/HOL for our development stems mostly from the benefits gained by adopting the HOL methodology (cf. Section 3.3).

The way to proceed in developing our geometry theory is very much in the spirit of Hilbert's *Grundlagen*: we show that there is a number system, say a field such as the hyperreals, associated with the geometry and reduce consistency of Isabelle's geometric theory to that of hyperreal arithmetic. This is readily achieved, when working within the context of Isabelle/HOL, by developing the geometry theory according to the HOL-methodology i.e. strictly through definitions that capture the notions (points, lines, signed areas, etc.) that are being dealt with and then proving that the various properties follow.

To carry out this task, the hyperreal theories of Isabelle are extended with the notions of hyperreal vectors. In essence, this is an algebraic approach which develops geometric objects and relations between these objects in the Cartesian product $\mathbb{R}^{*n}$ of the field of hyperreals, where $n = 2$. We have also developed a theory of vectors in three dimensions (and defined operations such as cross-products) but since this paper addresses geometry theorem proving in the plane, we shall only consider two dimensional vectors. The hyperreals are chosen rather than the reals since we can then express infinitesimal geometric notions as well. The definitions that are mechanized are given next— we start with a real vector theory which we then extend to get the hyperreal vectors and their algebraic operations.

5.1 Real Vector Space

In general, the simplest definition for a real vector in n dimensions is as an n-tuple of real numbers, $(r_1, \ldots, r_n)$. However, a more geometric definition can be provided that suits our purpose well.

Definition 5.1 *Given two points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ in $\mathbb{R}^2$, the vector $Q - P$ is called the directed line segment from P to Q. The components of the directed line segment are the terms in the pair $(x_2 - x_1, y_2 - y_1)$.*

In this definition, we implicitly assume that the origin is given by the real coordinates $(0, 0)$ and hence that a particular point is specified by the vector whose components correspond to its Cartesian coordinates. In Isabelle, we formulate a theory of two-dimensional vectors by first introducing vectors as a new type corresponding to a pair of real numbers:

$$\texttt{realv} \equiv \{\texttt{p} :: (\texttt{real} * \texttt{real}).\ \texttt{True}\}$$

As for the hyperreals, Isabelle automatically provides coercion functions — `Rep_realv` and its inverse `Abs_realv` in this case— that enable us to define basic operations on the new type. On a more intuitive level, one may simply read `Abs_realv` as:

$$\texttt{Abs_realv}\ (x, y) \equiv \begin{pmatrix} x \\ y \end{pmatrix}$$

in what follows.

We can then define the various operations on the new type. For example, the *inner product* or *dot product* of two vectors P and Q is defined, using tuples as patterns in abstractions [20], by:[3]

$$P \cdot Q \equiv \ (\lambda((x_1, y_1), (x_2, y_2)).\ x_1 x_2 + y_1 y_2) \ \ (\texttt{Rep_realv}\ P, \texttt{Rep_realv}\ Q)$$

This definition is slightly more complicated than the usual textbook one since it uses an explicit λ-abstraction and the representation function. However, we prove theorems that capture the more familiar definitions and which can then be fed to Isabelle's simplifier for rewriting. So, for the dot product, we have:

$$\texttt{Abs_realv}\ (x_1, y_1) \cdot \texttt{Abs_realv}\ (x_2, y_2) = x_1 x_2 + y_1 y_2$$

Similarly, we also define other important operations, such as *outer product* $(\times)$ and *scalar* multiplication $(\cdot_s)$. For clarity, we give their definitions as the simplification theorems proved in Isabelle rather than the actual definitions in terms of `Rep_realv` and λ-abstractions. The Isabelle definitions unfortunately tend to be slightly cluttered and become somewhat hard to read. So, for outer and scalar products we prove the following rules:

$$\texttt{Abs_realv}\ (x_1, y_1) \times \texttt{Abs_realv}\ (x_2, y_2) = x_1 y_2 - y_1 x_2$$

$$a \cdot_s \texttt{Abs_realv}\ (x, y) = \texttt{Abs_realv}\ (ax, ay)$$

[3] In what follows, the multiplication sign $(\cdot)$ between real variables is omitted whenever no ambiguity is likely to result.

For any two vectors P and Q, the outer product can be viewed as defining the oriented area of a parallelogram, with the vectors as two of the sides of the parallelogram. With this nice geometric interpretation in mind, the next step involves proving various properties of the outer product. The following theorem, which shows that the outer product is not commutative, is thus proved:

$$P \times Q = (-Q) \times P$$

Geometrically, this means a change in the orientation of the area while its magnitude remains unaffected. The negation of a vector P, for its part, is defined by negating its various components. In Isabelle:

$$-P \equiv (\lambda(x_1, x_2).\, \texttt{Abs_realv}\ (-x_1, -x_2))(\texttt{Rep_realv}\ P)$$

In the next section, the definition of signed area of a triangle follows directly from the geometric interpretation and algebraic behaviour associated with the outer product.

Various other algebraic properties of the operations introduced so far are proved in Isabelle. A few straightforward ones that are useful to the development are as follows:

$$\begin{array}{ll} u \cdot v = v \cdot u & u \cdot (v + w) = u \cdot v + u \cdot w \\ u \times u = 0 & (a \cdot_s u) \cdot (b \cdot_s v) = ab \cdot_s (u \cdot v) \\ u \cdot (u \times v) = 0 & (a \cdot_s u) \times (b \cdot_s v) = ab \cdot_s (u \times v) \end{array}$$

In these theorems, the *zero vector* is defined, as expected, by

$$0 \equiv \texttt{Abs_realv}\ (0, 0)$$

Another important concept that has not yet been introduced is that of the *length* or *norm* of a vector. For a vector P, this is defined by taking the square root of the dot product $P \cdot P$. In Isabelle,

$$\texttt{rvlen}\, P \equiv \texttt{sqrt}\ (P \cdot P)$$

The above definition is formalized directly but does rely on the square root operation and theorems about its properties being available in Isabelle. For instance, to prove that[4]

$$\texttt{rvlen}\ (k \cdot_s u) = \texttt{abs}\ k * \texttt{rvlen}\ u$$

the following theorem (with $x, y \in \mathbb{R}$) needs to be available in the theorem prover:

$$[|0 \le x; 0 \le y|] \Longrightarrow \texttt{sqrt}\ (x \cdot y) = \texttt{sqrt}\ x \cdot \texttt{sqrt}\ y$$

The existence of operations such as square root is often taken for granted in textbooks when new definitions depending on them are introduced. However, in a definitional mechanization such as ours, formalizing such concepts and their properties can sometimes result in a fair amount of work. In this particular case though, we benefit from our work on the mechanization of nonstandard real analysis [10, 14, 13]: this provides us with the square root operator and various theorems about it. Other important theorems proved in the theory include:

[4] In Isabelle, abs x denotes $|x|$.

- Cauchy-Schwarz inequality: $\texttt{abs}\ (u \cdot v) \leq \texttt{rvlen}\ u \cdot \texttt{rvlen}\ v$
- Minkowski inequality: $\texttt{rvlen}\ (u + v) \leq \texttt{rvlen}\ u + \texttt{rvlen}\ v$

After proving some further results of vector algebra, we develop a simple geometry theory based on the geometric interpretation of vectors and their operations. In the next sections, the definitions and results of the vector geometry development, as it currently stands, are outlined.

5.2 Real Vector Geometry

Chou, Gao, and Zhang have also used vector calculations in automated geometry theorem proving [2]. They assert a set of basic rules about the operations that can be carried out on vectors. Theorems are then derived using these basic axioms of the theory. The algorithm used by Chou et al. is nice and relatively simple: given a construction sequence for a geometric configuration, the points (i.e. vector variables) are eliminated one at a time from the vector expression standing for the conclusion, until only independent vector variables are left. The conclusion that results is then tested to see if it is identically zero.

In contrast to the above approach, we proceed by means of definitions only and having introduced real vectors and defined the operations on them, there is enough algebraic power for the theories to express geometric concepts: orthogonality and parallelism, signed (or oriented) areas, congruence of angles and much more. Moreover, we proceed mostly through simplification and substitution steps that are be applied to both the conclusion and premises of the current goal. That is, the proof steps in Isabelle are not limited to point elimination only.

We first introduce as basic geometric objects the notions of points and lines by defining the following types in Isabelle:

$$\texttt{pt} \equiv \{\texttt{p} :: \texttt{hypvec}.\ \texttt{True}\}$$
$$\texttt{line} \equiv \{\texttt{l} :: (\texttt{pt} * \texttt{pt}).\ \texttt{True}\}$$

From these definitions, a point is therefore specified by a position vector and a (directed) line given by a pair of vectors representing its end-points. These definitions give the theory a separate, nicer geometric interpretation in which geometric objects (points and lines) are dealt with rather than vectors of hyperreal numbers. The abstraction and representation functions of Isabelle enable us to deal with the underlying vector theory to prove basic properties of parallelism, perpendicularity, collinearity etc. Once this is done, we can hope to work at a higher abstract level which deals with geometric relations and interact rather minimally with the underlying vector constructions. This is similar in spirit with our construction of numbers, say the reals by Dedekind cuts, where initially for each operation we have to prove cut properties but as more theorems are proved, we deal less and less with the actual cuts and more with the algebra of the reals.

However, in the subsequent exposition we shall regard position vectors and points as being interchangeable when giving the definitions and describing properties proved. This abuse of notation is simply to make the definitions more readable on paper since it avoids the use of the coercion functions. We will show the definitions or theorems as actually formulated if the need ever arises. We also note that the notation A— B, used

in Isabelle for a line from point A to point B, is syntactic sugar for $\texttt{Abs_line}(A, B)$. Therefore, for each geometric condition, we have the corresponding vector definition:

1. That A, B, and C are collinear:

$$\texttt{coll}\, C\, A\, B \equiv (C - A) \times (B - A) = 0$$

2. That AB is parallel to CD:

$$A -\!- B \parallel C -\!- D \equiv (B - A) \times (D - C) = 0$$

3. That AB is perpendicular to CD:

$$A -\!- B \perp C -\!- D \equiv (B - A) \cdot (D - C) = 0$$

4. The length of a line AB:

$$\texttt{len}\, (A -\!- B) \equiv \texttt{rvlen}\, (B - A)$$

5. The signed area of triangle ABC:

$$\texttt{area}\; A\; B\; C \equiv 1/2 \cdot_s (B - A) \times (C - A)$$

6. The angle between AB and CD:

$$\langle A -\!- B, B -\!- C \rangle \equiv \texttt{arcos}\, (\texttt{unitv}\, (A - B) \cdot \texttt{unitv}\, (C - B))$$

   where

$$\texttt{unitv}\; P = (1/\texttt{rvlen}\; P) \cdot_s P$$

   The definition of the angle relies on the theory of transcendental functions developed in Isabelle. In our work on the formalization of analysis, the various trigonometric functions are defined over the reals through their power series expansions, and then extended to the hyperreals [14].

With these definitions set up, we verify that the basic properties of signed areas actually hold and justify the statements of geometric relations that were made by Chou et al. in terms of them [3]. The theorems about the sign of the area depending on the ordering of the vertices of the triangle are all proved automatically without any problems since our definition makes them direct consequences of the algebraic properties of the outer product. Consider, for example:

$$\begin{aligned} -\texttt{area}\; a\; c\; b &= -1/2 \cdot_s (c - a) \times (b - a) \\ &= -1/2 \cdot_s (-(b - a)) \times (c - a) \\ &= -\,-\,1/2 \cdot_s (b - a) \times (c - a) \\ &= \texttt{area}\; a\; b\; c \end{aligned}$$

Many similar rules are proved with the help of Isabelle's automatic tactic and added to the simplifier. The definition of parallelism in terms of signed areas, as given by Chou et al. [3], is also easily verified:

$$a -\!- b \parallel c -\!- d \Longleftrightarrow (\texttt{area}\; a\; b\; c = \texttt{area}\; a\; b\; d)$$

and the following theorem defining incidence (or collinearity) in terms of signed area:

$$\texttt{coll}\, a\, b\, c \Longleftrightarrow (\texttt{area}\ a\ b\ c = 0) \tag{5.1}$$

We also extend the definition of incidence to that of a set of points incident on a line, thereby enabling us to prove some more theorems. We can deal with the ratios of oriented lines by proving theorems such as these:

- $A -- B \parallel C -- D$ $(C \neq D)$:

$$\frac{\texttt{len}\,(A -- B)}{\texttt{len}\,(C -- D)} = \frac{(B - A) \cdot (D - C)}{(D - C) \cdot (D - C)}$$

- if R is the foot of the perpendicular from point A to line PQ $(P \neq Q)$:

$$\frac{\texttt{len}\,(P -- R)}{\texttt{len}\,(P -- Q)} = \frac{(A - P) \cdot (Q - P)}{\texttt{len}\,(P -- Q)^2}$$

- if two non-parallel lines intersect at a point R:

$$\texttt{len}\,(P -- R) \cdot (Q - P) \times (V - U) = \texttt{len}\,(P -- Q) \cdot (U - P) \times (V - U)$$

Some of the results above are unproved, high level lemmas stated by Chou et al. as being used in their automated GTP method based on vectors [2]. We verify all of them in Isabelle and store them as lemmas that become valuable when proving complicated geometry theorems. This verification of lemmas used in various established GTP methods is not a mere exercise as it supports the axiomatic geometry that we previously used in Isabelle for our mechanization of theorems from Newton's *Principia*[11, 10]. From a more general standpoint, it can also be viewed as provided a rigorous foundations for several automatic methods used in geometry theorem proving. Finally, since we are able to prove the expected geometric properties in the formalization, this gives us a relatively high degree of assurance that we are using the right definitions for various concepts.

5.3 Introducing the Infinitesimal Geometry

We start by defining the new type of hyperreal vectors using sequences of real vectors (i.e. essentially sequences of pairs of real numbers) and our free ultrafilter $U_{\mathbb{N}}$. As mentioned previously, the definitions are analogous to those used for defining the hyperreals. Once again, the various operations (e.g. dot product, outer product, addition, etc.) are defined in terms of pointwise operations on the underlying sequence (Fig. 1). Various properties, analogous to those of real vectors (see Section 5.1), are proved for the hyperreal vectors and their associated operations. All the mechanized proofs are straightforward as the properties follow directly from their real vector counterparts.

In addition, we distinguish between various types of vectors by means of their lengths. This characterization is analogous and closely related to that of the hyperreal numbers:

```
HyperVector = Transc +

constdefs

 (* equivalence relation *)
 hvrel "((nat ⇒ realv) * (nat ⇒ realv)) set"
 "hvrel ≡ {p. ∃ r s. p = (r,s) ∧ {n. r n = s n}∈U_ℕ}"

typedef
 hrealv ≡ "{x::(nat ⇒ realv). True}/hvrel"   (Equiv.quotient_def)

instance
 hrealv :: {zero, plus, minus}

defs
 hrealv_zero_def  "0 ≡ Abs_hrealv(hvrel^^{λn::nat. 0})"

constdefs
 (* norm can use nonstandard extension of square root operation *)
 hvlen :: hrealv ⇒ hypreal
 "hvlen u ≡ (*f* sqrt) (u · u)"

 hrealv_minus :: hrealv ⇒ hrealv
 "- P ≡ Abs_hrealv(⋃X∈Rep_hrealv(P). hyvrel^^{λn::nat. - (X n)})"

 (* embedding for the real vectors: use constant sequence *)
 hrealv_of_realv  :: realv ⇒ hrealv
 "hrealv_of_realv u ≡ Abs_hrealv(hvrel^^{λn::nat. u})"

 (* hyperreal unit vector *)
 hunitv :: hrealv ⇒ hrealv
 "hunitv u ≡ inv(hvlen u) ·_s u"

defs
 hrealv_add_def
 "P + Q  ≡ Abs_hrealv(⋃X∈Rep_hrealv(P).⋃Y∈Rep_hrealv(Q).
                       hvrel^^{λn::nat. X n + Y n})"

 hrealv_dot_def
 "P · Q  ≡ Abs_hrealv(⋃X∈Rep_hrealv(P).⋃Y∈Rep_hrealv(Q).
                       hvrel^^{λn::nat. X n · Y n})"
      ...

 hrealv_oprod_def
 "P × Q  ≡ Abs_hrealv(⋃X∈Rep_hrealv(P).⋃Y∈Rep_hrealv(Q).
                       hvrel^^{λn::nat. X n × Y n})"
```

FIG. 1. Isabelle/HOL theory for hyperreal vectors

Definition 5.2 *A hyperreal vector P is said to be infinitesimal, finite, or infinite if its length (*`hvlen` *P) is infinitesimal, finite, or infinite respectively. Moreover, P is infinitely close to Q ($P \approx_v Q$) if and only if $Q - P$ is infinitesimal.*

With this definition formalized in Isabelle, the following equivalence theorem about infinitely close vectors is proved:

$$[\langle X_n \rangle] \approx_v [\langle Y_n \rangle] \iff \begin{array}{l} [\langle \mathtt{fst}((X_n)) \rangle] \approx [\langle \mathtt{fst}((Y_n)) \rangle] \\ \wedge\ [\langle \mathtt{snd}((X_n)) \rangle] \approx [\langle \mathtt{snd}((Y_n)) \rangle] \end{array} \tag{5.2}$$

where $[\langle X_n \rangle]$ denotes the equivalence class of sequences of real vectors containing $\langle X_n \rangle$, and `fst` and `snd` are the first and second projection functions respectively provided by Isabelle for reasoning about pairs. The actual Isabelle theorem, though slightly overwhelming maybe, shows the relation between the various concepts explicitly and can be instructive:

$$\begin{array}{l} \mathtt{Abs_hrealv}\ (\mathtt{hvrel``}\{X\}) \approx_v \mathtt{Abs_hrealv}\ (\mathtt{hvrel``}\{Y\}) \\ \iff \mathtt{Abs_hypreal}\ (\mathtt{hyprel``}\{\lambda n.\, \mathtt{fst}(X_n)\}) \approx \\ \qquad \mathtt{Abs_hypreal}\ (\mathtt{hyprel``}\{\lambda n.\, \mathtt{fst}(Y_n)\}) \wedge \\ \qquad \mathtt{Abs_hypreal}\ (\mathtt{hyprel``}\{\lambda n.\, \mathtt{snd}(X_n)\}) \approx \\ \qquad \mathtt{Abs_hypreal}\ (\mathtt{hyprel``}\{\lambda n.\, \mathtt{snd}(Y_n)\}) \end{array}$$

In other words, two hyperreal vectors are infinitely close if and only if their components in corresponding positions are infinitely close to one another. This is a useful theorem that can be used in many cases to reduce infinitesimal reasoning involving hyperreal vectors to similar reasoning over the real vectors or even over the reals. We also prove the following important theorems about the different types of vectors:

1. P is infinitesimal if and only if all its components are infinitesimal.
2. P is finite if and only if all its components are finite.
3. P is infinite if and only if at least one of its components is infinite.

and many other interesting nonstandard theorems about the algebra of the operations and relations on them, such as:

$$a \in \mathtt{Finite} - \mathtt{Infinitesimal} \Longrightarrow (a \cdot_s w \approx_v a \cdot_s z) = (w \approx_v z) \tag{5.3}$$

$$c \in \mathtt{Finite} - \mathtt{Infinitesimal} \Longrightarrow (c \cdot_s w \approx_v b \cdot_s z) = (w \approx_v (b/c) \cdot_s z) \tag{5.4}$$

$$[|a \approx 0; u \in \mathtt{VFinite}|] \Longrightarrow a \cdot_s u \approx_v 0 \tag{5.5}$$

$$x \approx_v y \Longrightarrow \mathtt{hvlen}\, x \approx \mathtt{hvlen}\, y \tag{5.6}$$

$$u \in \mathtt{VFinite} - \mathtt{VInfinitesimal} \Longrightarrow u \cdot u \in \mathtt{Finite} - \mathtt{Infinitesimal} \tag{5.7}$$

$$u \in \mathtt{VFinite} - \mathtt{VInfinitesimal} \Longrightarrow (u \times v \approx 0) = (\exists k.\, v \approx_v k \cdot_s u) \tag{5.8}$$

where `VInfinitesimal` and `VFinite` denote the sets of infinitesimal and finite vectors respectively. Most of these theorems are relatively straightforward to prove although some like (5.6) and (5.8)are more challenging. We highlight some of the issues involved in their mechanization by examining part of the proof of theorem (5.8) more closely. At first sight, one might expect the proof of the theorem to be similar to that of:

$$u \neq 0 \Longrightarrow (u \times v = 0) = (\exists k.\, v = k \cdot_s u) \tag{5.9}$$

which is easily proved in Isabelle by unfolding the definitions of the various vector operations and then reducing the reasoning to equation solving. However, a similar approach in which we unfold $\approx_v$ using (5.2)and then try to prove the theorem by reasoning over the hyperreals is much harder. This is because the infinitely close relation ($\approx$), unlike equality, is not closed under multiplication and goals involving multiplication and $\approx$ require a lot of work (case-splits) to be established. Our mechanization, therefore goes for a direct approach involving reasoning over hyperreal vectors and their operations. We will only consider the (trickier) first part of the proof which involves showing that:

$$[|u \in \texttt{VFinite} - \texttt{VInfinitesimal}; u \times v \approx 0|] \Longrightarrow \exists k.\, v \approx_v k \cdot_s u \tag{5.10}$$

For the mechanization of this goal, after some experimentation, we decide to define the following operation on real vectors:

$$\texttt{ortho}\,(\texttt{Abs_realv}\,(x_1, y_1)) = \texttt{Abs_realv}\,(-y_1, x_1)$$

with the following nonstandard extension to hyperreal vectors:

$$\texttt{hortho}\,[\langle X_n \rangle] \equiv [\langle \texttt{ortho}\, X_n \rangle]$$

Geometrically, the operation can be viewed as defining a new vector orthogonal (perpendicular) to the given one. Using these definitions, we then easily prove by simplification the following theorems:

$$(v \times w) \cdot_s \texttt{hortho}\, u = (u \cdot v) \cdot_s w - (u \cdot w) \cdot_s v \tag{5.11}$$

$$u \in \texttt{VFinite} \Longrightarrow \texttt{hortho}\, u \in \texttt{VFinite} \tag{5.12}$$

As a brief remark, we note that theorem (5.11)can be viewed as a lower dimensional (planar) analogy of the spatial triple vector product $u \times (v \times w)$ (in which vector $v \times w$, for example, then denotes the so-called cross product). Now, using theorems (5.5)and (5.12)with the assumptions of conjecture (5.10),we derive:

$$(u \times v) \cdot_s \texttt{hortho}\, u \approx_v 0$$

which, using theorem (5.11),rewrites to:

$$(u \cdot u) \cdot_s v \approx_v (u \cdot v) \cdot_s u \tag{5.13}$$

From the first assumption of goal (5.10)and theorem (5.7), we have:

$$u \cdot u \Longrightarrow \texttt{Finite} - \texttt{Infinitesimal}$$

Using this and theorem (5.4),we derive that

$$v \approx_v ((u \cdot v)/(u \cdot u)) \cdot_s u$$

from which the conclusion of (5.10)follows immediately. This overview demonstrates, we hope, the somewhat intricate nature of proofs involving nonstandard concepts. Although the statement of the theorem (5.10)is very similar to that of theorem (5.9), the actual proofs are very different. The infinitely close relation introduces numerous

subtleties that one might overlook were it not for the strict definitional framework of Isabelle/HOL. In particular, the care that must be exercised when multiplying two infinitely close quantities (e.g. $a \approx_v b$) with some other quantity (say c) to ensure that the results are also infinitely close ($c \cdot_s a \approx c \cdot_s b$) is never allowed to lapse. The proof that we have just outlined, though relatively easy to understand, is not an immediately obvious one; its mechanization required a fair amount of thought and subsequent experimentation in Isabelle.

As a final note on this proof, we remark that the operator `ortho` (and hence `hortho`) is not a concept that we considered when developing the initial vector theory. It was defined during the mechanization of theorem (5.10)to simplify the proof. Subsequently, however, we realised that it had many nice properties, such as $u \times v = \texttt{ortho}\, u \cdot v$, $\texttt{ortho}\,(u+v) = \texttt{ortho}\, u + \texttt{ortho}\, v$, and $u \cdot \texttt{ortho}\, u = 0$ amongst others. This highlights how the mechanization of a particular theorem can lead to the definition of new concepts which further enrich the theory.

The nonstandard vector theorems, we believe, have clear geometric readings and formalize the intuitive behaviour one would expect. Theorem (5.6),for example, can

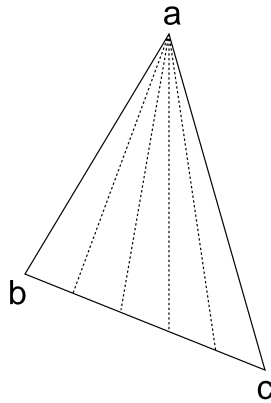


FIG. 2. A "shrinking" triangle

be used directly to prove an intuitive theorem about a shrinking triangle in which one of the sides is infinitesimal. In Fig. 2, for example, one can intuitively see that as the length of bc becomes smaller, the lengths of ab and ac approach each other, until they are infinitely close when bc is infinitesimal. This is captured by the following Isabelle theorem:

$$\texttt{len}\,(b -\!\!- c) \approx 0 \Longrightarrow \texttt{len}\,(a -\!\!- b) \approx \texttt{len}\,(a -\!\!- c)$$

Interestingly, if the lengths of the sides ab and bc are *real* valued, then they have to be *equal* (i.e. triangle abc is an isosceles) when bc is infinitesimal:

$$[|\texttt{len}\,(a -\!\!- b) \in \mathbb{R}; \texttt{len}\,(b -\!\!- c) \in \mathbb{R}; \texttt{len}\,(a -\!\!- c) \approx 0|]$$
$$\Longrightarrow \texttt{len}\,(a -\!\!- b) = \texttt{len}\,(b -\!\!- c)$$

This is because of theorem (4.2) stating that two real numbers that are infinitely close to one another are effectively equal. We also formally derive, for example, theorems

such as:

$$[|\texttt{len}\ (a \mathrel{--} b) \in \texttt{Finite}; \texttt{len}\ (b \mathrel{--} c) \in \texttt{Infinitesimal}|] \Longrightarrow \texttt{area}\ a\ b\ c \approx 0$$

and

$$[|\texttt{coll}\ a\, b\, c; \texttt{area}\ p\ b\ c \approx 0|] \Longrightarrow \texttt{area}\ p\ a\ c \approx \texttt{area}\ p\ a\ b \qquad (5.14)$$

The latter (see Fig. 3) is proved using the cancellation theorem (5.3),as well as various others involving associativity and commutativity of vector addition to perform AC-rewriting. These are just a few of the infinitesimal geometry theorems involving

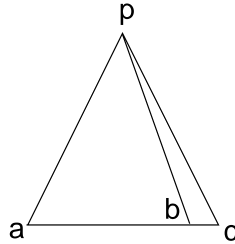


FIG. 3. Infinitely close areas

familiar geometric concepts. We next introduce a number of basic concepts systematically defined using the various notions from our nonstandard vector theory.

6 Some Infinitesimal Geometric Notions

Each of the new definitions can be viewed as weakening of the more familiar ones. We start with a nonstandard formulation of parallelism and orthogonality.

Almost parallel and almost perpendicular

Just as the concept of two lines being parallel was introduced, using hyperreal vectors the weaker notion of two lines being *almost parallel* is defined (with $A \neq B$ and $C \neq D$):

$$\begin{aligned} A \mathrel{--} B \parallel_a C \mathrel{--} D \equiv\ & \texttt{hunitv}\ (B - A) \approx_v \texttt{hunitv}\ (D - C)\ \vee \\ & \texttt{hunitv}\ (B - A) \approx_v -\texttt{hunitv}\ (D - C) \end{aligned}$$

We trivially prove that this is an equivalence relation. More importantly, the relation between this definition and that of parallel lines, given in Section 5.2, is highlighted by the following theorem, also proved in Isabelle:

$$\begin{aligned} & D - C \in \texttt{VFinite} - \texttt{VInfinitesimal} \\ & \Longrightarrow A \mathrel{--} B \parallel_a C \mathrel{--} D \Longleftrightarrow (B - A) \times (D - C) \approx 0 \end{aligned} \qquad (6.1)$$

The theorem expresses the almost parallel property in a form similar to that of ordinary parallelism, with equality replaced by the infinitely close relation. However, there is a notable difference which is shown as an additional condition on one of the

two lines (CD in this particular case, although it could have been on AB since $\|_a$ is symmetric). Without the condition, (6.1) above is not a theorem as the outer product of an infinitesimal and infinite vector is not necessarily infinitely close to zero. Also, in terms of area, justifying a more geometrically intuitive definition based on signed areas, we have:

$$\begin{aligned}&\texttt{len}\ (C \;\text{--}\; D) \in \texttt{Finite} - \texttt{Infinitesimal} \Longrightarrow \\ &A \;\text{--}\; B \parallel_a C \;\text{--}\; D \Longleftrightarrow (\texttt{area}\ a\ c\ d \approx \texttt{area}\ b\ c\ d)\end{aligned}$$

We also define the notion of two lines being almost perpendicular. Once again, we make use of the notion of unit vector to get a suitable definition. Lines

$$A \;\text{--}\; B \perp_a C \;\text{--}\; D \equiv \texttt{hunitv}\ (B - A) \cdot \texttt{hunitv}\ (D - C) \approx 0$$

We note that since the dot product produces a hyperreal, we use the infinitely close relation $\approx$ over these numbers rather than $\approx_v$ which is defined over hyperreal vectors.

Almost collinear

We next introduce the notion of three points being *almost collinear*. Intuitively, one might expect three points a, b, and c to be almost collinear (denoted by $\texttt{acoll}\ a\, b\, c$ in Isabelle) if and only if the signed area $\texttt{area}\ a\ b\ c$ is infinitely close to zero. Such a definition would be very similar in spirit to the equivalence theorem (5.1).However, since our geometry allows both infinitesimal and infinite quantities, this definition is inadequate: it does not hold in the case where two of the points concerned, say b and c, are infinitely far apart and the third one, say a, is infinitely close to the line bc. This is because the outer product $(c - b) \times (a - b)$ is not necessarily infinitely close to zero in this case as well. Instead, we define the property as follows:

$$\texttt{acoll}\ a\, b\, c \equiv (b - a) \parallel_a (b - c)$$

and prove a number of theorems involving it such as the variant of (5.14),shown in Fig. 4:

$$\begin{aligned}&[|\texttt{len}\ (b \;\text{--}\; a) \in \texttt{Finite} - \texttt{Infinitesimal}; \texttt{acoll}\ a\, b\, c; \texttt{area}\ p\ b\ c \approx 0|] \\ &\Longrightarrow \texttt{area}\ p\ a\ c \approx \texttt{area}\ p\ a\ b\end{aligned}$$

Infinitesimal angles

Our NSA theory is powerful enough to prove theorems involving the trigonometric functions and infinitesimal angles. For example, we can formally formulate and prove assertions such as

$$\sin(\theta) = \theta \text{ and } \cos(\theta) = 1 \;\; \textit{where } \theta \textit{ is infinitely small}$$

that one often sees in textbooks. These are rarely given any further justification: the reader needs to rely on her knowledge of trigonometric functions and on her intuition about what infinitely small means to see that the statements are indeed plausible.

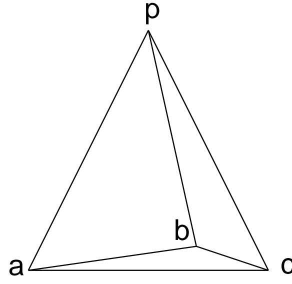


FIG. 4. Infinitely close areas

Such assertions can be formalized in NSA, however, by making θ an infinitesimal and replacing equality by the infinitely close relation $\approx$. The proofs are intuitive, yet rigorous, and relatively easy to mechanize. We give, as an example, a brief proof of the statement $\sin^*(\theta) \approx \theta$.

In the NSA theory [13] of Isabelle/HOL, the formal nonstandard definition of the derivative of a function f at x (DERIV) is given by:

$$\mathsf{DERIV}(x)\ f :> d \equiv \forall h \in \mathsf{Infinitesimal} - \{0\}.\ \frac{f^*(\underline{x} + h) - \underline{\underline{f(x)}}}{h} \approx \underline{d}$$

This is simply saying that the derivative of f at x is d if $\frac{\Delta f}{\Delta x}$ is *infinitely close* to d. With this, and assuming the standard results (proved in Isabelle) that

$$\cos(0) = 1, \sin(0) = 0,$$

and

$$\mathsf{DERIV}(x)\ (\lambda x.\ \sin(x)) :> \cos(x),$$

we can easily prove that $\sin^*(\theta) \approx \theta$ for all infinitesimal θ.

Proof:

if $\theta = 0$: This is trivial since $\approx$ is reflexive.
else if $\theta \neq 0$: Since $\texttt{DERIV}(x)\ \lambda x.\ \sin(x) :> \cos(x)$, for all x, we have that

$$\begin{aligned}
&\texttt{DERIV}(0)\ \lambda x.\ \sin(x) :> \cos(0)\\
&\Rightarrow \forall h \in \texttt{Infinitesimal} - \{0\}.\ \frac{\sin^*(0 + h) - \underline{\underline{\sin(0)}}}{h} \approx 1\\
&\Rightarrow \frac{\sin^*(0 + \theta) - \underline{\underline{\sin(0)}}}{\theta} \approx 1\\
&\Rightarrow \frac{\sin^*(\theta)}{\theta} \approx 1\\
&\Rightarrow \sin^*(\theta) \approx \theta
\end{aligned}$$

As a remark, we note that we have used theorem (4.4) and a theorem stating that $\texttt{Infinitesimal} \subseteq \texttt{Finite}$ to reach the final step. Through a similar reasoning, we also prove that $\cos^*(\theta) \approx 1$ and, interestingly, that $\tan^*(\pi/2 + \theta) \in \texttt{Infinite}$, for

all infinitesimal θ. We expect such results involving angles and trigonometry will to prove useful in the further development of the geometry.

In addition, we also prove that the angle between two lines which are almost perpendicular is infinitely close to $\pi/2$, i.e.,

$$a -\!\!- b \perp_a c -\!\!- d \Longleftrightarrow \langle a -\!\!- b, c -\!\!- d \rangle \approx \pi/2$$

Almost similar triangles

This is basically the notion of *ultimately similar* triangles that we have described and used a number of times before [11, 12]. We briefly recall its definition here:

$$\begin{array}{lcl} \texttt{USIM}\ a\ b\ c\ a'\ b'\ c' & \equiv & \langle b -\!\!- a, a -\!\!- c \rangle \approx \langle b' -\!\!- a', a' -\!\!- c' \rangle \ \wedge \\ & & \langle a -\!\!- c, c -\!\!- b \rangle \approx \langle a' -\!\!- c', c' -\!\!- b' \rangle \ \wedge \\ & & \langle c -\!\!- b, b -\!\!- a \rangle \approx \langle c' -\!\!- b', b' -\!\!- a' \rangle \end{array}$$

We are still formally investigating the properties of this concept. We have already reproduced in our new setting most of the theorems described in previous work [12]. Similarly, we have defined the notion of two triangles being almost congruent.

7 Nonstandard Proofs of Standard Geometry Theorems

Our nonstandard technique is strong enough to produce nice proofs of traditional geometry theorems. We consider, as a short case study, a nonstandard proof that the area of a circle of radius r is πr^2. The area of the circle will be shown to be infinitely close to the area of an enclosed (inscribed) polygon with infinitely many sides. The exact real value area can then be obtained by taking the standard part of this polygonal area.

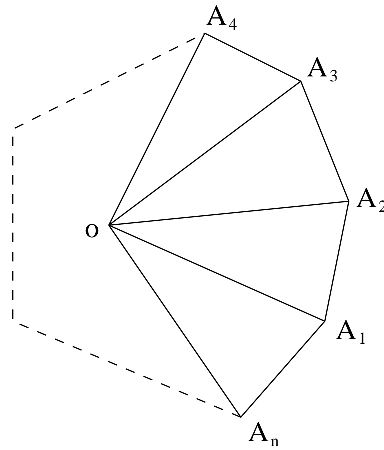


FIG. 5. A closed polygon

In Fig. 5, the area of the closed polygon $A_1 \ldots A_n$ is defined by the formula:

$$\text{area } A_1 \ldots A_n \equiv OA_1A_2 + OA_2A_3 + \ldots + OA_{n-1}A_n + OA_nA_1$$

where OA_1A_2, for example, represents the area of triangle OA_1A_2 which was defined in Section 5.2 in terms of the vector outer-product in the plane. The value of the polygonal area is independent of O but depends on OA_i, the radius vector to the ith point. The definition of polygonal area looks recursive except for the last area term (OA_nA_1) and so, in Isabelle, motivates the following formalization with the zero vector as the origin O:

$$\texttt{polyArea} :: (\texttt{nat} \Rightarrow \texttt{real} \Rightarrow \texttt{realv}) \Rightarrow \texttt{nat} \Rightarrow \texttt{real} \Rightarrow \texttt{real}$$
$$\texttt{polyArea}\ P\ n\ r \equiv \texttt{pArea}\ P\ n + \texttt{area}\ 0\ (P\,n\,r)\ (P\,0\,r)$$

and the following primitive recursive definition for the area from A_0 to A_{n-1}:

$$\texttt{pArea} :: (\texttt{nat} \Rightarrow \texttt{real} \Rightarrow \texttt{realv}) \Rightarrow \texttt{nat} \Rightarrow \texttt{real} \Rightarrow \texttt{real}$$
$$\texttt{primrec}$$
$$\texttt{pArea}\ P\ 0\ r = 0$$
$$\texttt{pArea}\ P\ (\texttt{Suc}\ n)\ r = \texttt{pArea}\ P\ n\ r + \texttt{area}\ 0\ (P\,n\,r)\ (P\,(\texttt{Suc}\ n)\,r)$$

Thus, according to our definition, the polygon is defined as a sequence of functions from reals to real vectors. The real value r acts as a parameter which can be used to determine the ith point. This is needed as often the radius vectors OA_i does not depend on just i but also on some other quantity such as an angle. The two parameters (e.g. multiplied) together enable us to progress along the curve being approximated. An alternative way of looking at the polygon is to consider each radius vector as being given by $A_{r(i)}$. This means that we could probably specify the definitions above without the fixed parameter r being given explicitly— it would be part of the definition of the polygon. However, one possible advantage of our chosen formalization is that we can have a general definition for the inscribed polygon (see (7.1), for example) which specifies the angle as a argument to be supplied.

Now, if $\mathcal{C}$ is a circle of radius 1, for example, we can inscribe a polygon $A_1 \cdots A_n$ by choosing points A_1, A_2,..., A_n in order along it. If n is an infinite hypernatural number then the points A_i crowd one another, and we expect to arrive at the formula for the area enclosed by $\mathcal{C}$. We call such a polygon an *hyperfinite polygon*.

Our definition `polyArea`, however, is purely standard and can only consider the area of polygons with increasingly large but *still* finite (natural) number of points. We therefore extend the definition to deal with polygons with a hypernatural i.e. nonstandard number of points. This is defined as follows:

$$\texttt{hpolyArea} :: (\texttt{nat} \Rightarrow \texttt{real} \Rightarrow \texttt{realv}) \Rightarrow \texttt{hypnat} \Rightarrow \texttt{hypreal} \Rightarrow \texttt{hypreal}$$
$$\texttt{hpolyArea}\ PNR \equiv \texttt{Abs_hypreal}(\bigcup X \in \texttt{Rep_hypnat}\ N. \bigcup Y \in \texttt{Rep_hypreal}\ R. \texttt{hyprel``}\{\lambda n.\,\texttt{polyArea}\ P\ (Xn)\ (Yn)\})$$

or, equivalently, without the coercion functions:

$$\texttt{hpolyArea}\ P\ [\langle X_n\rangle]\ [\langle Y_n\rangle] \equiv [\langle \texttt{polyArea}\ P\ X_n\ Y_n\rangle]$$

With this defined, we can now see how to determine the area of the circle using our infinite polygonal approximation technique.

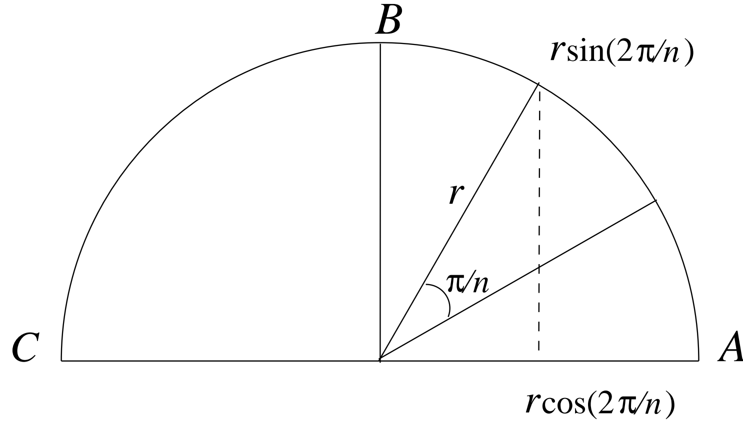


FIG. 6. Inscribing a polygon of n sides in a semi-circle

In our mechanized proof, we first consider the unit semi-circle ABC (see Fig. 6). Using the angle θ between successive radius vectors as parameter, the polygon can be defined by the following sequence of real vectors:

$$\lambda k\,\theta.\,\texttt{Abs_realv}\;(\cos k\theta, \sin k\theta) \tag{7.1}$$

where k denotes the k-th point of the polygon. Hence, given that $n \in \mathbb{N}$ points are inscribed in the semi-circle, the angle between the radius vectors is π/n and so the polygonal area is denoted by:

$$\texttt{polyArea}\;(\lambda k\,\theta.\,\texttt{Abs_realv}\;(\cos k\theta, \sin k\theta))\;n\;(\pi/n)$$

We then easily prove by induction and with the help of the mechanized lemma:

$$\sin(x-y) = \cos y \sin x - \sin y \cos x$$

supplied to Isabelle's simplifier that the following theorem holds:

$$\texttt{polyArea}\;(\lambda k\,\theta.\,\texttt{Abs_realv}\;(\cos k\theta, \sin k\theta))\;n\;(\pi/n) = 1/2n\sin(\pi/n) \tag{7.2}$$

We also prove the following property of polygonal areas:

$$\texttt{polyArea}\;(\lambda nr.\,c \cdot_s P\;n\;r)\;N\;R = c^2 \cdot \texttt{polyArea}\;P\;N\;R$$

which means that for a semi-circle of radius $r \in \mathbb{R}$, we have:

$$\texttt{polyArea}\;(\lambda k\,\theta.\,\texttt{Abs_realv}\;(r\cos k\theta, r\sin k\theta))\;n\;(\pi/n) = 1/2r^2 n\sin(\pi/n) \tag{7.3}$$

Now, if $n = [\langle X_m \rangle]$, the number of inscribed points, is an infinite hypernatural number, we have that π/n is infinitesimal. But, from the result in the previous section about infinitesimal angles, we know that:[5]

$$\frac{\sin^*(\pi/n)}{(\pi/n)} \approx 1$$

[5] For clarity, we omit to show the embedding functions `hypreal_of_hypnat` and `hypreal_of_real` used to embed the hypernatural number n and π respectively in the hyperreals.

and hence that

$$n \sin^*(\pi/n) \approx \pi$$

This result, with (7.3) above, allows us to prove that:

$$\texttt{hpolyArea}\,(\lambda k\,\theta.\,\texttt{Abs_realv}\;(r\cos k\theta, r\sin k\theta))\;n\;(\pi/n) \approx 1/2\pi r^2$$

since

$$\begin{aligned}
&\texttt{hpolyArea}\,(\lambda k\,\theta.\,\texttt{Abs_realv}\;(r\cos k\theta, r\sin k\theta))\;n\;(\pi/n)\\
&= \texttt{hpolyArea}\,(\lambda k\,\theta.\,\texttt{Abs_realv}\;(r\cos k\theta, r\sin k\theta))\;[\langle X_m\rangle]\;(\pi/[\langle X_m\rangle])\\
&= [\langle \texttt{polyArea}\,(\lambda k\,\theta.\,\texttt{Abs_realv}\;(r\cos k\theta, r\sin k\theta))\;X_m\;(\pi/X_m)\rangle]\\
&= [\langle 1/2r^2(X_m)\sin(\pi/X_m)\rangle]\\
&= 1/2r^2[\langle X_m\rangle]\sin^*(\pi/[\langle X_m\rangle]) = 1/2r^2 n\; sin^*(\pi/n)
\end{aligned}$$

From this result, we deduce that for a circle, with the angle between successive radius vectors given by $2\pi/n$, the following holds:

$$\texttt{hpolyArea}\,(\lambda k\,\theta.\,\texttt{Abs_realv}\;(r\cos k\theta, r\sin k\theta))\;n\;(2\pi/n) \approx \pi r^2$$

Hence, by "exhausting" the circle with an inscribed polygon of infinite number of sides, we have formalized a nice, geometrically intuitive, proof that the area of the circle of radius r, is infinitely close to πr^2. In fact, if we assume that the area of the circle is real, then by the standard part theorem, it is equal to πr^2. We may get this behaviour directly by defining the polygonal area that we want (call it `PolyArea`) as the real quantity equal to the standard part of the hyperfinite polygonal area `hpolyArea` i.e.,

$$\texttt{PolyArea}\;P\,n\,r = \texttt{str}\;(\texttt{hpolyArea}\;P\,n\,r)$$

This means that our infinite approximation can, in effect, provide an exact real quantity for the area that we are exhausting. As a final note, we remark that most of the theorems just described are proved with a high degree of automation. Theorem (7.2), for example, is proved in two steps: induction on n followed by a call to one of Isabelle's automatic tactic.

8 Further Work

This paper has described some of our current work on the formalization and investigation of a geometry that rigorously admits both infinitesimal and infinite notions. We still have much of the geometry to explore though: one currently unproved conjecture, for example, is that two (co-planar) lines which are almost parallel do *meet* at a point infinitely far away i.e., we expect to have a well-defined, non-degenerate solution to the problem.

As a by-product of this work, we now have a relatively well developed vector theory in Isabelle. This contains many of the familiar theorems about vector operations as well as the new theorems involving the infinitely close relation, infinitesimal and infinite vectors, and other nonstandard concepts. As the work proceeds, we expect to add more theorems to provide a theory that can be useful for other purposes (e.g. proofs in mechanics that often involve vectors and as well as infinitesimals).

We will be introducing and investigating other, perhaps less obvious, *almost relations*. For example, we have recently mechanized notions of approximate geometric objects in which an ellipse with infinitely close foci, for instance, can be regarded as being almost (but not quite) a circle. Other notions include "almost betweenness", approximate point inclusion in a triangle, and "almost a tangent" to a circle, for example.

We will pursue our mechanization of geometric proofs that use infinitesimal and infinite quantities to reach infinitely accurate approximation results. We have introduced the inductive notion of area for a closed polygon which can be used to approximate any closed figure (curve). Our example showed how this can be used to derive a relatively simple proof about the area of the circle using infinite numbers and infinitesimals. Our approach rigorously mechanizes the informal argument that one might give for such an proof. A standard proof of the same result, however, would have required us to introduce sequential limit arguments and then deal with the alternating quantifiers [10, 13].

Our geometric techniques capture well the ideas embodied in proofs that use the "Method of Exhaustion" of Archimedes. In these, one figure is usually approximated more and more accurately by another one in order to compute geometric quantities such as boundaries, areas, and volumes. This sort of reasoning, however, cannot be dealt with by existing (standard) mechanical geometry theorem proving methods. The work of Baron [1], for example, provides a wealth of such proofs throughout the centuries for us to work with and mechanize.

9 Concluding Remarks

In this paper, we have formally introduced the notion of an infinitesimal geometry based on hyperreal vectors. We have briefly sketched some aspects relating this hyperreal geometry to non-Archimedean geometry. Various theorems have been proved that have no direct counterparts in Euclidean geometry since the latter only deals with real numbers.

Vector algebra offers an attractive approach to mechanical geometry theorem proving. There is much active research going on using the related field of Clifford algebra, which is generally regarded as being more expressive [22, 8]. In our case, since we are doing interactive rather than automatic theorem proving, vectors provide a simple and adequate approach to analytic geometry. Also, as was shown by Dieudonné, inner (dot) and outer products of vectors are sufficient to develop elementary geometry [7].

As far as we are aware, this is the first mechanization of a theory of hyperreal vectors. Moreover, Keisler's textbook is, to our knowledge, the only work to give a brief exposition of a vector theory [18]. As a result, most of the theorems mechanized in Isabelle have been proved independently of any previous work or textbooks. We have shown that these vectors obey the usual algebraic rules for vectors since they form an inner product space over the field $\mathbb{R}^*$. By using the extended vectors instead of real vectors, it is possible to describe, in addition to ordinary geometric concepts, the novel notions of infinitesimal geometry presented in this paper.

The analytic geometry development was carried out to provide a rigorous definitional approach in which to investigate our infinitesimal geometry. By following the HOL methodology, we have the guarantee that our formalization is consistent and

that all results proved are actual theorems about the geometry we have developed.

We also remark on an important realisation emphasized by the current work: the inclusion of infinitesimals and other nonstandard concepts in geometry introduces subtle issues that can easily lead to inadequate definitions. Indeed, it can be problematic to formulate concepts that rely on some form of product (outer, dot, multiplication etc.) as the operation can be ill-defined whenever it involves both an infinitesimal and an infinite quantity. We became especially aware of the subtlety involved when our initial definition for almost parallel lines (we used the equivalence theorem (6.1)without the associated condition) proved inadequate. We could not prove some of the properties we felt should hold since we were implicitly ruling out an infinitesimal line and an infinite line being almost parallel.

The realisation came after some experimentation with the framework and did force us to exercise much more care. However, the fact that we encountered such a problem is probably unsurprising. After all, the flaw that we found in one of the famous proofs of the great Newton was also of this nature [12]; it involved taking the ill-defined product of an infinitesimal and an infinite quantity. This is a useful experience that will help us as we explore more challenging concepts in this geometry.

Acknowledgement

This research was funded by ESPRC grant GR/M45030 'Computational Modelling of Mathematical Reasoning'. I would like to thank the anonymous referees for their insightful comments.

References

[1] M. E. Baron. *The Origins of the Infinitesimal Calculus*. Pergammon Press, 1969.

[2] S. C. Chou, X. S. Gao, and J. Z. Zhang. Automated geometry theorem proving by vector calculation. In *ACM-ISSAC*, pages 284–291, Kiev Ukraine, July 1993.

[3] S. C. Chou, X. S. Gao, and J. Z. Zhang. Automated generation of readable proofs with geometric invariants, I. multiple and shortest proof generation. *Journal of Automated Reasoning*, 17:325–347, 1996.

[4] S. C. Chou, X. S. Gao, and J. Z. Zhang. Automated generation of readable proofs with geometric invariants, II. theorem proving with full-angles. *Journal of Automated Reasoning*, 17:349–370, 1996.

[5] A. Church. A formulation of the simple theory of type. *Journal of Symbolic Logic*, 5:56–68, 1940.

[6] P. J. Davis and R. Hersh. *The Mathematical Experience*. Harmondsworth, Penguin, 1983.

[7] J. Dieudonné. *Linear Algebra and Geometry*. Hermann, 1969. Translated from the original French text *Algèbre linéaire et géométrie élémentaire*.

[8] S. Fevre and D. Wang. Proving geometric theorems using clifford algebra and rewrite rules. In C Kirchner and H. Kirchner, editors, *Automated Deduction – CADE-15*, volume 1421 of *Lecture Notes in Artificial Intelligence*, pages 17–32. Springer-Verlag, July 1998.

[9] G. Fisher. Veronese's non-Archimedean linear continuum. In P. Ehrlich, editor, *Real Numbers, Generalizations of the Reals, and Theories of Continua*, volume 242 of *Synthese Library*. Kluwer Academic Publisher, 1994.

[10] J. D. Fleuriot. *A combination of geometry theorem proving and nonstandard analysis, with application to Newton's Principia*. PhD thesis, Computer Laboratory, University of Cambridge, 1999. Available as Computer Laboratory Technical Report 469.

[11] J. D. Fleuriot and L. C. Paulson. A combination of geometry theorem proving and nonstandard analysis, with application to Newton's *Principia*. In C Kirchner and H. Kirchner, editors,

Automated Deduction – CADE-15, volume 1421 of *Lecture Notes in Artificial Intelligence*, pages 3–16. Springer-Verlag, July 1998.

[12] J. D. Fleuriot and L. C. Paulson. Proving Newton's Propositio Kepleriana using geometry and nonstandard analysis in Isabelle. In X.-S. Gao, D. Wang, and L. Yang, editors, *Automated Deduction in Geometry*, volume 1669 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1999.

[13] J. D. Fleuriot and L. C. Paulson. Mechanizing nonstandard real analysis. *LMS Journal of Computation and Mathematics*, 3:140–190, 2000.

[14] Jacques D. Fleuriot. On the mechanization of real analysis in Isabelle/HOL. In J. Harrison and M. Aagaard, editors, *Theorem Proving in Higher Order Logics: 13th International Conference, TPHOLs 2000*, volume 1869 of *Lecture Notes in Computer Science*, pages 146–162. Springer-Verlag, 2000.

[15] M. Gordon and T. Melham. *Introduction to HOL: A theorem proving environment for Higher Order Logic*. Cambridge University Press, 1993.

[16] D. Hilbert. *The Foundations of Geometry*. The Open Court Company, 1901. Translation by E. J. Townsend.

[17] A. E. Hurd and P. A. Loeb. *An Introduction to Nonstandard Real Analysis*, volume 118 of *Pure and Applied Mathematics*. Academic Press Inc., 1985.

[18] H. J. Keisler. *Foundations of Infinitesimal Calculus*. Prindle, Weber & Schmidt, 1976.

[19] L. C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer, 1994.

[20] L. C. Paulson. Isabelle's object-logics. Technical Report 286, Computer Laboratory, University of Cambridge, February 1998.

[21] H. Poincaré. Review of Hilbert's foundations of geometry (1902). In P. Ehrlich, editor, *Real Numbers, Generalizations of the Reals, and Theories of Continua*, volume 242 of *Synthese Library*. Kluwer Academic Publisher, 1994.

[22] D. Wang. Clifford algebraic calculus for geometric reasoning, with application to computer vision. In D. Wang, R. Caferra, L. Fariñas del Cerro, and H. Shi, editors, *Automated Deduction in Geometry, ADG'96*, volume 1360 of *Lecture Notes in Artificial Intelligence*, pages 115–140. Springer, 1997.

Received September 1, 2000. Revised: December 1, 2000, January 19, 2001

A Simple Formalization and Proof for the Mutilated Chess Board

LAWRENCE C. PAULSON, *Computer Laboratory, University of Cambridge, England, E-mail: lcp@cl.cam.ac.uk.*

Abstract

The impossibility of tiling the mutilated chess board has been formalized and verified using Isabelle. The formalization is concise because it is expressed using inductive definitions. The proofs are straightforward except for some lemmas concerning finite cardinalities. This exercise is an object lesson in choosing a good formalization: one at the right level of abstraction.

Keywords: mutilated chess board, inductive definitions, Isabelle

1 Introduction

A chess board can be tiled by 32 dominoes, each covering two squares. If two diagonally opposite squares are removed, can the remaining 62 squares be tiled by dominoes? No. Each domino covers a white square and a black square, so a tiled area must have equal numbers of both colours. The mutilated board cannot be tiled because the two removed squares have the same colour (Fig. 1).

The mutilated chess board problem has stood as a challenge to the automated reasoning community since McCarthy [8] posed it in 1964. Robinson [15] outlines the history of the problem, citing Max Black as its originator.

Anybody can grasp the argument instantly, but even formalizing the problem seems hard, let alone proving it. McCarthy has recently renewed his challenge, publishing a formalization that he claims is suitable for any 'heavy duty set theory' prover [9].

Formalizations like this destroy the simplicity of the original problem. They typically define complicated predicates to recognize objects. To recognize dominoes, a predicate checks whether its argument contains two adjacent squares. Subramanian defines *adjacent* by comparing co-ordinates [17, 18]:

```
(defn adjp (s1 s2)
  (or (and (equal (car s1) (car s2))
           (equal (plus 1 (cdr s1)) (cdr s2)))
      (and (equal (cdr s1) (cdr s2))
           (equal (plus 1 (car s1)) (car s2)))
      ))
```

Subramanian makes other definitions whose combined effect is to recognize a list of non-overlapping dominoes and to compute the region covered. McCarthy's formalization has a similar flavour, though posed in the language of sets. It is concise but formidable.

An alternative is to express the notion of tiling by an inductive definition. It is concise and nearly as clear as the informal problem statement. It provides an induction principle that is well-suited to proving the desired theorem.

 ©Oxford University Press

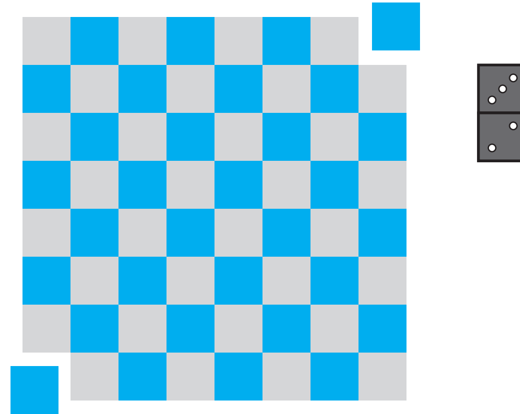


FIG. 1. The Mutilated Chess Board

2 Mathematical development

First we must make the intuitive argument rigorous. A *tile* is a set, regarded as a set of positions. A *tiling* (using a given set A of tiles) is defined inductively to be either the empty set or the union of a tiling with a tile $a \in A$ disjoint from it. Thus, a tiling is a finite union of disjoint tiles drawn from A.

This view is abstract and general. None of the sets have to be finite; we need not specify what positions are allowed. Now let us focus on chess boards.

A *square* is a pair (i, j) of natural numbers: an *even* (or *white*) square if $i + j$ is even and otherwise an *odd* (or *black*) square.

Let $\mathsf{lessThan}(n) = \{i \mid i < n\}$. (In set theory $n = \{i \mid i < n\}$ by definition, but some people find that confusing.) The Cartesian product $\mathsf{lessThan}(8) \times \mathsf{lessThan}(8)$ expresses a 64-square chess board; it is the union of 8 disjoint rows of the form $\{i\} \times \mathsf{lessThan}(8)$ for $i = 0, \ldots, 7$.

A *domino* is a tile of the form $\{(i, j), (i, j + 1)\}$ or $\{(i, j), (i + 1, j)\}$. Since tilings are finite, we can use induction to prove that every tiling using dominoes has equally many even squares as odd squares.

Every row of the form $\{i\} \times \mathsf{lessThan}(2n)$ can be tiled using dominoes. As the union of two disjoint tilings is itself a tiling, every matrix of the form $\mathsf{lessThan}(2m) \times \mathsf{lessThan}(2n)$ can be tiled using dominoes. So every $2m \times 2n$ matrix has as many even squares as odd squares. (Informal treatments never bother to prove that a chess board has equal numbers of black and white squares.) The diagonally opposite squares $(0, 0)$ and $(2m - 1, 2n - 1)$ are both even; removing them results in a set that has fewer even squares than odd squares. No such set, including the mutilated chess board, can be tiled using dominoes.

3 The formal definitions

Isabelle [12] is a generic proof assistant, supporting many logics including ZF set theory and higher-order logic. I have done this exercise using both Isabelle/ZF and Isa-

```
Mutil = Main +

consts     tiling :: "('a set) set ⇒ ('a set) set"
inductive "tiling A"
  intrs
    empty  "{} ∈ tiling A"
    Un     "⟦a ∈ A;  t ∈ tiling A;  a ∩ t = {}⟧
            ⟹ a ∪ t ∈ tiling A"

consts     domino  :: "(nat*nat)set set"
inductive "domino"
  intrs
    horiz  "{(i, j), (i, Suc j)} ∈ domino"
    vertl  "{(i, j), (Suc i, j)} ∈ domino"

constdefs
  coloured  :: "nat ⇒ (nat*nat)set"
   "coloured b == {(i,j). (i+j) mod #2 = b}"

end
```

FIG. 2. Isabelle/HOL Definitions of Dominoes and Tilings

belle/HOL. The definitions and proofs are similar in both systems. My formalization should be easy to mechanize in theorem provers that support inductive definitions, such as Coq [4] and HOL [5]. Higher-order logic simplifies the presentation slightly; type checking eliminates premises such as $i \in \mathsf{nat}$.

Figure 2 presents the theory file for the Isabelle/HOL version. It makes all the definitions needed for the chess board problem: tilings, dominoes and square colourings. Note that `Suc` is the successor function (mapping n to $n+1$) and that `#2` denotes the number two. Keywords of the theory file syntax are underlined for clarity.

An inductive definition specifies the desired introduction rules. An Isabelle package defines the appropriate least fixedpoint and proves the introduction and induction rules [11]. The set of tilings using a set A of tiles is defined inductively. The Isabelle syntax appearing in Fig 2 expresses these two rules:

$$\emptyset \in \mathsf{tiling}(A) \qquad \frac{a \in A \quad t \in \mathsf{tiling}(A) \quad a \cap t = \emptyset}{a \cup t \in \mathsf{tiling}(A)}$$

Why does `tiling` have type `('a set)set ⇒ ('a set)set`? The symbol `'a` is a *type variable*. Isabelle/HOL is *polymorphic*: the type-checker automatically replaces each type variable by the type required by the context. In effect, `'a` is the type of squares. Each tile is a set of squares, so it has type `'a set`. The set A of tiles therefore has type `('a set)set`, as does the set of tilings generated by A.

The set of dominoes is inductively defined too. The Isabelle syntax expresses two introduction rules:

$$\{(i,j),(i,j+1)\} \in \mathsf{domino} \qquad \{(i,j),(i+1,j)\} \in \mathsf{domino}$$

The 'induction' here is trivial, but no matter, this definition is easy to use. It is declarative. Contrast it with the version appearing in Sect. 1, which is a piece of Lisp

code. The constant `domino` has type `(nat*nat)set set` because it is a set of sets of pairs of natural numbers.

Figure 2 defines coloured b as set of squares having colour b. Formally, it is the set of even squares if $b = 0$ and the odd squares if $b = 1$. The set $\mathsf{lessThan}(n)$ is predefined in Isabelle/HOL to be $\{i \mid i < n\}$.

4 A primer on rule induction

You are probably familiar with 'mathematical induction' and with structural induction over lists and similar datatypes. An inductive definition gives rise to a principle sometimes known as *rule induction*. Given the definition of tiling, Isabelle generates the corresponding induction rule, shown here using mathematical notation:

$$\frac{z \in \mathsf{tiling}(A) \qquad P(\emptyset) \qquad \begin{matrix}[a \in A \quad t \in \mathsf{tiling}(A) \quad P(t) \quad a \cap t = \emptyset] \\ \vdots \\ P(a \cup t)\end{matrix}}{P(z)}$$

In English, *a property P that is closed under the introduction rules for* $\mathsf{tiling}(A)$ *holds for all elements of* $\mathsf{tiling}(A)$. Induction is sound because $\mathsf{tiling}(A)$ is the *least* set closed under those rules. (This is why it is called rule induction.) In the inductive step, we are given an arbitrary tile $a \in A$ and tiling $t \in \mathsf{tiling}(A)$ that are disjoint ($a \cap t = \emptyset$) and satisfy the induction hypothesis $P(t)$.

A trivial rule induction proves that if each $a \in A$ is a finite set then so is $\mathsf{tiling}(A)$. Here $P(z)$ is the property $\mathsf{finite}(z)$. By induction, it suffices to show

- $\mathsf{finite}(\emptyset)$, which is trivial,
- and that $a \in A$ and $\mathsf{finite}(t)$ imply $\mathsf{finite}(a \cup t)$. This holds because we have assumed $\mathsf{finite}(a)$ for all $a \in A$.

The induction rule for dominoes has no induction hypothesis. A property holds for all dominoes provided it holds for the two possibilities given in the inductive definition. In the last two premises, i and j are arbitrary natural numbers.

$$\frac{z \in \mathsf{domino} \quad P(\{(i,j),(i,j+1)\}) \quad P(\{(i,j),(i+1,j)\})}{P(z)}$$

It is time for a harder example of induction. Let us prove that the union of two disjoint tilings is itself a tiling:

$$\frac{t \in \mathsf{tiling}(A) \quad u \in \mathsf{tiling}(A) \quad t \cap u = \emptyset}{t \cup u \in \mathsf{tiling}(A)}$$

This induction must be set up with care. Here $P(z)$ is the formula

$$u \in \mathsf{tiling}(A) \to (t \cap u = \emptyset \to t \cup u \in \mathsf{tiling}(A)) \qquad (4.1)$$

The induction formula must be an implication because the induction variable, t, also occurs in $t \cap u = \emptyset$.

By induction on t there are two cases.

```
Goal "t ∈ tiling A ⟹
      u ∈ tiling A → t ∩ u = {} → t ∪ u ∈ tiling A";
by (etac tiling.induct 1);
by (simp_tac (simpset() addsimps [Un_assoc]) 2);
by Auto_tac;
```

FIG. 3. Isabelle/HOL Proof: the Union of Disjoint Tilings is a Tiling

- *Base case.* Putting $t = \emptyset$ in the formula (4.1), we must show

$$u \in \mathsf{tiling}(A) \to (\emptyset \cap u = \emptyset \to \emptyset \cup u \in \mathsf{tiling}(A))$$

  This is trivial because $\emptyset \cup u = u \in \mathsf{tiling}(A)$.

- *Inductive step.* We assume disjoint sets $a \in A$ and $t \in \mathsf{tiling}(A)$, as usual. The induction hypothesis is simply (4.1). We must show

$$u \in \mathsf{tiling}(A) \to ((a \cup t) \cap u = \emptyset \to (a \cup t) \cup u \in \mathsf{tiling}(A))$$

  To prove this implication, we assume $u \in \mathsf{tiling}(A)$ and $(a \cup t) \cap u = \emptyset$, which yields $a \cap u = \emptyset$ and $t \cap u = \emptyset$. From the induction hypothesis (4.1) we have $t \cup u \in \mathsf{tiling}(A)$. Since a is disjoint from both t and u, we may add it to the tiling $t \cup u$ to obtain $a \cup (t \cup u) \in \mathsf{tiling}(A)$.

5 The mechanical proofs

The Isabelle proofs offer few surprises. Finite cardinalities are tricky to reason about, as I have noted in previous work [14]. I needed a couple of hours to find a machine proof that a domino consists of one even square and one odd square. Another trouble spot was to prove that removing elements from a finite set reduces its cardinality: $|A - \{x\}| < |A|$ if A is finite and $x \in A$. One outcome of this exercise is a collection of general theorems about remainders and cardinality, which I have installed in Isabelle/HOL.

Apart from these trouble spots, the mechanized proof was straightforward. Developing the original ZF version took under 24 working hours. Excluding facts added to libraries, the (HOL) definitions and proof script occupy about 4400 bytes. They execute in 8.5 seconds on a 600MHz Pentium. Both figures are tiny, as suits this toy problem.

Figure 3 presents part of the script: the inductive proof outlined in the previous section. The script may be difficult to understand, but we see that proving this theorem requires little detail from the user. The `Goal` command supplies the theorem to be proved. The next line applies rule induction. Then the simplifier (`simp_tac`) is called with an associativity theorem in order to replace $(a \cup t) \cup u$ by $a \cup (t \cup u)$. The rest of the proof is done by the automatic proof tactic, `Auto_tac`.

The full proof script, comprising 13 theorems, is Appendix A. Isabelle can display formulas using the fonts of X-symbol package [19], making formulas much more readable on-screen than they are in raw ASCII; I have edited the script to use similar symbols. Let us review the proofs informally.

5.1 On tiling chess boards

The first theorem has already been discussed in Sect. 4 and Fig. 3. We now develop a geometry of chess boards. The next two theorems (each proved by `Auto_tac`) relate `lessThan(Suc n)` and Cartesian products.

```
lessThan(Suc n) × B  =  ({n} × B) ∪ ((lessThan n) × B)
A × lessThan(Suc n)  =  (A × {n}) ∪ (A × (lessThan n))
```

Next comes a lemma, proved by `Auto_tac`, concerning singleton sets and Cartesian products. It makes a useful rewrite rule.

```
({i} × {n}) ∪ ({i} × {m}) = {(i,m), (i,n)}
```

The next two results state that *a row or matrix with an even number of columns can be tiled with dominoes.*

```
        {i} × lessThan(#2*n) ∈ tiling domino
(lessThan m) × lessThan(#2*n) ∈ tiling domino
```

These theorems apply to a standard 8×8 chess board, but not to a 9×9 one. The first theorem has a four-step proof, by induction on `n`. The simplifier massages `lessThan(#2 * Suc n)` into the union of a domino with the tiling given in the induction hypothesis. Then a tiling rule is applied explicitly. Finally, the automatic tactic (given the lemma proved above) finishes off. The second theorem has a trivial proof: induction over `m` followed by `Auto_tac`.

5.2 On colours and dominoes

Here is a simple fact about the squares in a tiling of a specified colour.

```
coloured b ∩ (insert (i,j) t) =
  (if (i+j) mod #2 = b then insert (i,j) (coloured b ∩ t)
                       else coloured b ∩ t)
```

Here $\mathsf{insert}\, x\, A$ denotes $\{x\} \cup A$. The b-coloured squares of $\{(i,j)\} \cup t$ comprise the b-coloured squares of t along with (i,j), if this square is coloured b. Although obvious, this fact is useful for rewriting. The proof is a one-liner: `Auto_tac`.

This fact is used to prove that a domino covers one square of each colour:

```
d ∈ domino ⟹
    (∃i j. coloured 0 ∩ d = {(i,j)}) ∧
    (∃m n. coloured 1 ∩ d = {(m,n)})
```

The proof is again simple. The first step is induction (really case analysis) on the domino. The automatic tactic finishes the proof, given a rewrite rule that reduces $(m+1) \bmod n$ to $m \bmod n$.

5.3 On the cardinalities of some finite sets

For us, a domino is a two-element set of squares. Clearly all dominoes are finite, and a region tiled by dominoes is finite. Both proofs use induction followed by `Auto_tac`.

```
d ∈ domino         ⟹ finite d
t ∈ tiling domino ⟹ finite t
```

Most of the papers describing the chess board proof omit to mention that the board has finitely many squares. However, finiteness is crucial to the counting argument. (Infinite tiling problems are very different from finite ones. An infinite chess board can be tiled with dominoes even after one black square has been removed.)

Every set tiled by dominoes (such as an 8×8 chess board) contains equally many black squares as white ones. Here `card` is the cardinality function.

```
t ∈ tiling domino ⟹ card(coloured 0 ∩ t) = card(coloured 1 ∩ t)
```

This fact is also usually omitted from informal accounts, presumably because it is obvious. But its proof, six steps long, is not trivial. After applying induction, we use a fact proved above, namely that a domino covers one square of each colour. We are left having to show

```
card(insert sq0 (coloured 0 ∩ t)) = card(insert sq1 (coloured 1 ∩ t))
```

where `sq0` and `sq1` are the newly covered squares. The induction hypothesis is

```
card(coloured 0 ∩ t) = card(coloured 1 ∩ t).
```

Two proof steps show that the uses of `insert` add a square that was not already in the set. The result follows because both cardinalities increase by one.

5.4 Towards the main result

The main result presents some difficulties. Take the general case of removing any two white (even) squares, not necessarily in the corners.

```
⟦ t ∈ tiling domino;
  (i+j) mod #2 = 0;  (m+n) mod #2 = 0;
  {(i,j),(m,n)} ⊆ t ⟧
⟹ (t - {(i,j)} - {(m,n)}) ∉ tiling domino
```

In English, *removing two white squares from a region tiled with dominoes leaves a region that cannot be tiled.* The proof consists of five steps. The first simply assumes that the region can be tiled, for contradiction. Next we claim that there are fewer white squares than black, from which (step 3) we immediately obtain a contradiction. The last two steps prove the claim. It is surprisingly hard to prove that removing two elements from the set of white squares reduces its cardinality.

The main result is proved for any board with positive even dimensions. The mutilated board (less the two corners) cannot be tiled with dominoes.

```
t = lessThan(#2 * Suc m) × lessThan(#2 * Suc n)
⟹ t - {(0,0)} - {(Suc(#2*m), Suc(#2*n))} ∉ tiling domino
```

The proof applies the general theorem just discussed and discharges the first subgoal using a tiling lemma proved in Sect. 5.1. The rest falls to `Auto_tac`.

6 Related work and conclusions

In this note there is no space for a full literature review. Several efforts [2, 16, 18] are in the same spirit as the present work: the chess board is formalized and impossibility

of tiling proved following the intuitive argument about colours. Other work has used exhaustive search or radical reformulations of the problem.

The Isabelle formalization compares favourably with the others. The definitions (Fig. 2) are concise, and in my view, easy to understand. The script is short: under 120 lines compared with over 500 for Subramanian [17]. (In terms of characters, which is more accurate, the ratio drops to 1:3.) According to McCarthy [9], Bancerek's mechanization [2] in Mizar requires 400 lines. Rudnicki's version [16] (also in Mizar) requires 300 lines. Andrews [1] reports a complex proof; it is not clear how much effort is needed to generate it.

When are inductive definitions appropriate? The choice is partly a matter of taste; published formalizations of the mutilated chess board show great diversity. Inductive definitions are ideal for finite constructions that allow non-determinism; the laying down of tiles fits that description precisely. The inductive definition plays the same role as Subramanian's finite state machine [18]. The initial state is the empty board; next states are obtained by adding disjoint tiles; properties that hold of all reachable states are proved by induction. Giving an illegal input to the state machine sends it to an error state — a concept usually avoided with inductive definitions, since they describe only the legal constructions.

The finite state machine approach that Subramanian describes has been applied to substantial system verifications [10]. The inductive approach described above is an effective means of verifying cryptographic protocols [13]. Inductive definitions scale up to serious problems.

Acknowledgements. I learned of the expressiveness of inductive definitions through participation in the ESPRIT project 6453 TYPES, and especially through the work of Gérard Huet [6, 7]. John Harrison and anonymous referees commented on this paper.

References

[1] Peter B. Andrews and Matthew Bishop. On sets, types, fixed points, and checkerboards. In Pierangelo Miglioli, Ugo Moscato, Daniele Mundici, and Mario Ornaghi, editors, *Theorem Proving with Analytic Tableaux and Related Methods: 5th international workshop, TABLEAUX '96*, LNAI 1071, pages 1–15. Springer, 1996.

[2] Grzegorz Bancerek. The mutilated chessboard problem — checked by Mizar. In Boyer and Trybulec [3].

[3] Robert Boyer and Andrzej Trybulec, editors. *QED Workshop II.* On the World Wide Web at `http://www.mcs.anl.gov/qed/`, 1995.

[4] The Coq proof assistant. `http://coq.inria.fr/`, 2000.

[5] M. J. C. Gordon and T. F. Melham. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic.* Cambridge University Press, 1993.

[6] Gérard Huet. The Gallina specification language : A case study. In *Proceedings of 12th FST/TCS Conference, New Delhi*, LNCS 652. Springer, 1992.

[7] Gérard Huet. Residual theory in λ-calculus: A formal development. *Journal of Functional Programming*, 4(3):371–394, 1994.

[8] John McCarthy. A tough nut for proof procedures. Memo 16, Stanford Artificial Intelligence Project, July 1964.

[9] John McCarthy. The mutilated checkerboard in set theory. In Boyer and Trybulec [3].

[10] J Strother Moore. *Piton: A Mechanically Verified Assembly-Level Language.* Kluwer Academic Publishers, 1996.

[11] Lawrence C. Paulson. A fixedpoint approach to implementing (co)inductive definitions. In Alan Bundy, editor, *Automated Deduction — CADE-12 International Conference*, LNAI 814, pages 148–161. Springer, 1994.

[12] Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover.* Springer, 1994. LNCS 828.

[13] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.

[14] Lawrence C. Paulson and Krzysztof Grąbczewski. Mechanizing set theory: Cardinal arithmetic and the axiom of choice. *Journal of Automated Reasoning*, 17(3):291–323, December 1996.

[15] J. A. Robinson. Formal and informal proofs. In Robert S. Boyer, editor, *Automated Reasoning: Essays in Honor of Woody Bledsoe*, pages 267–281. Kluwer Academic Publishers, 1991.

[16] Piotr Rudnicki. The mutilated checkerboard problem in the lightweight set theory of Mizar. `http://web.cs.ualberta.ca/~piotr/Mizar/Mutcheck`, November 1995.

[17] Sakthi Subramanian. A mechanically checked proof of the mutilated checkerboard theorem. `ftp://ftp.cs.utexas.edu/pub/boyer/nqthm/nqthm-1992/examples/subramanian/`, 1994.

[18] Sakthi Subramanian. An interactive solution to the $n \times n$ mutilated checkerboard problem. *Journal of Logic and Computation*, 6(4):573–598, 1996.

[19] Christoph Wedler. Emacs package "x-symbol": Overview. `http://www.fmi.uni-passau.de/~wedler/x-symbol/`, 2000.

A Full proof script

```
(*
The Mutilated Chess Board Problem, formalized inductively
*)

Addsimps (tiling.intrs @ domino.intrs);
AddIs    tiling.intrs;
```

Material discussed in Sect. 5.1

```
(** The union of two disjoint tilings is a tiling **)

Goal "t∈ tiling A ⟹ u∈ tiling A → t ∩ u = {} → t ∪ u ∈ tiling A";
by (etac tiling.induct 1);
by (simp_tac (simpset() addsimps [Un_assoc]) 2);
by Auto_tac;
qed_spec_mp "tiling_UnI";

AddIs [tiling_UnI];

(*** Chess boards ***)

Goalw [lessThan_def]
    "lessThan(Suc n) × B = ({n} × B) ∪ ((lessThan n) × B)";
by Auto_tac;
qed "Sigma_Suc1";

Goalw [lessThan_def]
    "A × lessThan(Suc n) = (A × {n}) ∪ (A × (lessThan n))";
by Auto_tac;
qed "Sigma_Suc2";

Addsimps [Sigma_Suc1, Sigma_Suc2];

Goal "({i} × {n}) ∪ ({i} × {m}) = {(i,m), (i,n)}";
```

```
by Auto_tac;
qed "sing_Times_lemma";

Goal "{i} × lessThan(#2*n) ∈ tiling domino";
by (induct_tac "n" 1);
by (ALLGOALS (asm_simp_tac (simpset() addsimps [Un_assoc RS sym])));
by (rtac tiling.Un 1);
by (auto_tac (claset(), simpset() addsimps [sing_Times_lemma]));
qed "dominoes_tile_row";

AddSIs [dominoes_tile_row];

Goal "(lessThan m) × lessThan(#2*n) ∈ tiling domino";
by (induct_tac "m" 1);
by Auto_tac;
qed "dominoes_tile_matrix";
```

Material discussed in Sect. 5.2

```
(*** "coloured" and Dominoes ***)

Goalw [coloured_def]
    "coloured b ∩ (insert (i,j) t) =
       (if (i+j) mod #2 = b then insert (i,j) (coloured b ∩ t)
                            else coloured b ∩ t)";
by Auto_tac;
qed "coloured_insert";
Addsimps [coloured_insert];

Goal "d ∈ domino ⟹ (∃i j. coloured 0 ∩ d = {(i,j)}) &
                     (∃m n. coloured 1 ∩ d = {(m,n)})";
by (etac domino.elim 1);
by (auto_tac (claset(), simpset() addsimps [mod_Suc]));
qed "domino_singletons";
```

Material discussed in Sect. 5.4

```
Goal "d ∈ domino ⟹ finite d";
by (etac domino.elim 1);
by Auto_tac;
qed "domino_finite";
Addsimps [domino_finite];


(*** Tilings of dominoes ***)

Goal "t ∈ tiling domino ⟹ finite t";
by (etac tiling.induct 1);
by Auto_tac;
qed "tiling_domino_finite";

Addsimps [tiling_domino_finite, Int_Un_distrib, Diff_Int_distrib];

Goal "t ∈ tiling domino ⟹ card(coloured 0 ∩ t) = card(coloured 1 ∩ t)";
by (etac tiling.induct 1);
by (dtac domino_singletons 2);
```

```
by Auto_tac;
(*this lemma tells us that both "inserts" are non-trivial*)
by (subgoal_tac "∀p C. C ∩ a = {p} → p ∉ t" 1);
by (Asm_simp_tac 1);
by (Blast_tac 1);
qed "tiling_domino_0_1";
```

Material discussed in Sect. 5.3

```
(*Final argument is surprisingly complex*)
Goal "⟦ t ∈ tiling domino;
       (i+j) mod #2 = 0;  (m+n) mod #2 = 0;
       {(i,j),(m,n)} ⊆ t ⟧
     ⟹ (t - {(i,j)} - {(m,n)}) ∉ tiling domino";
by (rtac notI 1);
by (subgoal_tac "card (coloured 0 ∩ (t - {(i,j)} - {(m,n)})) <
               card (coloured 1 ∩ (t - {(i,j)} - {(m,n)}))" 1);
by (force_tac (claset(), HOL_ss addsimps [tiling_domino_0_1]) 1);
by (asm_simp_tac (simpset() addsimps [tiling_domino_0_1 RS sym]) 1);
by (asm_full_simp_tac
    (simpset() addsimps [coloured_def, card_Diff2_less]) 1);
qed "gen_mutil_not_tiling";

(*Apply the general theorem to the well-known case*)
Goal "t = lessThan(#2 * Suc m) × lessThan(#2 * Suc n)
     ⟹ t - {(0,0)} - {(Suc(#2*m), Suc(#2*n))} ∉ tiling domino";
by (rtac gen_mutil_not_tiling 1);
by (blast_tac (claset() addSIs [dominoes_tile_matrix]) 1);
by Auto_tac;
qed "mutil_not_tiling";
```

Received 11 September 2000. Revised: November 8, 2000, January 15, 2001

Acknowledgements

The Editor-in-Chief and the editor of this special issue would like to thank the following colleagues who have helped maintain the standards set for a scientific journal, through their refereeing of the papers that have been submitted.[1]

Roel Bloo
Carsten Butz
Thérèse Hardin
Daniel Hirschkoff
Patrik Holt
Michael Kohlhase
Tobias Nipkow
Nicola Olivetti
Vincent van Oostrom
Christine Paulin-Mohring
Randy Pollack
Femke van Raamsdonk
Nick Taylor.

[1] The list includes the referees for the papers in this issue, plus the referees of papers rejected meanwhile.

Interest Group in Pure and Applied Logics (IGPL)

The Interest Group in Pure and Applied Logics (IGPL) is sponsored by The European Association for Logic, Language and Information (FoLLI), and currently has a membership of over a thousand researchers in various aspects of logic (symbolic, mathematical, computational, philosophical, etc.) from all over the world (currently, more than 50 countries). Our main activity is that of a research and information clearing house.

Our activities include:

- Exchanging information about research problems, references and common interest among group members, and among different communities in pure and applied logic.
- Helping to obtain photocopies of papers to colleagues (under the appropriate copyright restrictions), especially where there may be difficulties of access.
- Supplying review copies of books through the journals on which some of us are editors.
- Helping to organise exchange visits and workshops among members.
- Advising on papers for publication.
- Editing and distributing a Newsletter and a Journal (the first scientific journal on logic which is FULLY electronic: submission, refereeing, revising, typesetting, publishing, distribution; first issue: July 1993): the Logic Journal of the Interest Group on Pure and Applied Logics. (For more information on the Logic Journal of the IGPL, see the Web homepage: http://www.jigpal.oupjournals.org)
- Keeping a public archive of papers, abstracts, etc., accessible via ftp.
- Wherever possible, obtaining reductions on group (6 or more) purchases of logic books from publishers.

If you are interested, please send your details (name, postal address, phone, fax, e-mail address, research interests) to:

IGPL Headquarters
c/o Prof. Dov Gabbay
King's College, Dept of Computer Science
Strand
London WC2R 2LS
United Kingdom
e-mail: dg@dcs.kcl.ac.uk

For the organisation, Dov Gabbay, Ruy de Queiroz and Hans Jürgen Ohlbach