SOLVING DOMAIN EQUATIONS in NaDSet

by
Paul C. Gilmore
and
George K. Tsiknis

Technical Report 91-31
December 1991

Department of Computer Science
University of British Columbia
Rm 333 - 6356 Agricultural Road
Vancouver, B.C.
CANADA V6T 1Z2

e-mail address: <gilmore@cs.ubc.ca>, <tsiknis@cs.ubc.ca>

SOLVING DOMAIN EQUATIONS
in
NaDSet
by

Paul C Gilmore & George K Tsiknis

Technical Report TR 91-31
December, 1991

Submission to Seventh Annual IEEE Sympoium on Logic in Computer Science
June 22-25, 1992, Santa Cruz, California

Abstract

Solutions of systems of domain equations is the basis for what is called the Strachey-Scott approach to providing a denotational semantics for programming languages. The solutions offered by the mathematical methods developed by Scott, however, do not provide a computational basis for the semantics in the form of a proof theory. The purpose of this paper is to provide such a theory using the logic and set theory NaDSet.

The development of NaDSet was motivated by the following three principles:

1) Abstraction, along with truth functions and quantification, is one of the three fundamental concepts of logic and should be formalized in the same manner as the other two.

2) Natural deduction presentations of logic provide a transparent formalization of Tarski's reductionist semantics.

3) Atomic formulas receive their truth values from a nominalist interpretation.

That these three principles lead to a successful resolution of the set theoretic paradoxes and to a sound formulation of NaDSet has been demonstrated elsewhere with proofs of consistency and completeness. Applications of NaDSet to programming language semantics, category theory, non-well-founded sets, and to foundational questions of mathematics have also been demonstrated.

The financial support of the Natural Science and Engineering Research Council of Canada is gratefully acknowledged.

1. INTRODUCTION

Solutions of systems of domain equations is the basis for what is called the Strachey-Scott approach to providing a denotational semantics for programming languages [Stoy77]. As described in [Gordon79], complex domains are formed from primitive domains such as Id, the identifiers, and Bv, the basic values, and from defined finite domains using four constructors, namely, disjoint sum [D+R], Cartesian product [D×R], sequences D*, and function space [D→R]. A system of domain equations takes the form:

$$D_1 = T_1 [D_1, \ldots, D_n]$$
$$D_2 = T_2 [D_1, \ldots, D_n]$$
$$\ldots$$
$$D_n = T_n [D_1, \ldots, D_n]$$

where each $T_i [D_1, \ldots, D_n]$ is an expression constructed out of variables $D_1, \ldots, D_n$ and finite and primitive domains using the domain constructors.

The solutions offered by the mathematical methods developed by Scott are not formal in the strictest sense of that word; that is, they do not provide a proof theory for denotational semantics. Indeed, Scott in his introduction to [Stoy77] noted: "For the future the problems of an adequate proof theory and of explaining non-determinism loom very large." The purpose of this paper is to provide such a proof theory within NaDSet. An important advantage of the proof theory provided here is that induction principles for least set definitions need not be added as assumptions; they are derivable from the definitions themselves. This was a major theme of [Gilmore&Tsiknis90a]. A further advantage may be that a modified proof theory may assist in "explaining non-determinism"; but the pursuit of that possibility requires additional study.

The logic and set theory NaDSet is sketched in §2, with a detailed treatment of its proof theory postponed for presentation by example. In §3 formal definitions of the domain constructors and related concepts are given in NaDSet, while in §4 systems of domain equations are defined together with their NaDSet solutions. Derivations for some of the results cited in §3 and §4 are given in §5. At the same time, the proof theory for NaDSet is explained with examples. The paper concludes in §6 with a discussion of programming semantics and the differences between Scott and NaDSet solutions to domain equations.

2. NaDSet

2.1. Motivation

NaDSet is a logic and set theory motivated by the following three principles:

1) Abstraction, along with truth functions and quantification, is one of the three fundamental concepts of logic and should be formalized in the same manner as the other two.
2) Natural deduction presentations of logic provide a transparent formalization of Tarski's reductionist semantics.
3) Atomic formulas receive their truth values from a nominalist interpretation.

That these three principles lead to a successful resolution of the set theoretic paradoxes and to a sound formulation of NaDSet is demonstrated in [Gilmore91a] where proofs of consistency and completeness are provided. Applications of NaDSet are demonstrated in [Gilmore89,91c], [Gilmore&Tsiknis91a,91b,92], and in [Tsiknis91], while an informal description of the logic is provided in [Gilmore91b]. Different formulations of NaDSet are described in [Gilmore&Tsiknis91c], while an earlier weaker version is described in [Gilmore80,86].

2.2. Abstraction Terms & Semantics

In order to formalize abstraction in a natural deduction presentation, abstraction terms such as $\{x \mid x=x\}$, $\{w \mid \exists u \exists v\ w=\langle u,v\rangle\}$, $\{w \mid \exists u\ (\sim w=u \wedge u:u)\}$, and $\{x \mid \sim x:x\}$ are admitted in NaDSet. The latter two terms illustrate the use of ':' in place of the conventional '$\in$' in NaDSet; the last term, for example, denotes Russell's paradoxical set. In the first two abstraction terms '=' is intensional identity, and in the second term $\langle u,v\rangle$ denotes the ordered pair of u and v. Both of these are defined below.

Semantic rules for these abstraction terms have the same character as Tarski's semantic rules for the logical connectives and quantifiers: The truth value of a nonatomic formula is dependent upon the truth values of simpler formulas, while the truth values of the irreducible atomic formulas are assumed given. Examples of irreducible atomic formulas are $\{x \mid x=x\}$:C, a:C, and C:C, where 'a' is a first order constant used as a name for a given object, while 'C' is a second order constant that names a set of objects. Interpretations of these formulas are conventional in the following sense: Given a domain of discourse **d**, an interpretation assigns an object in **d** to the terms '$\{x \mid x=x\}$', 'a', and 'C' appearing to the left of ':' and a subset of **d** to the constant 'C' appearing to the right. The formula C:C is true in the interpretation, for example, if the member of **d** assigned to the occurrence of 'C' to the left of ':' is a member of the subset of **d** assigned to the occurrence of 'C' to the right.

The nominalist character of this interpretation arises from a restriction of the membership of **d** to terms that serve as names of objects or of sets; sets themselves cannot be members. NaDSet has distinct notations for free and bound variables, where variables may be bound in abstraction terms as well as by quantifiers. Free variables are called parameters. **d** consists of the terms without occurrences of parameters.

The nominalist interpretation of the atomic formulas has an important consequence for the parameters of NaDSet. An interpretation can assign to a parameter either a member of **d** or a subset of **d**; to avoid an abuse of use and mention, the same parameter cannot be assigned to both. Thus the nominalist interpretation of its atomic formulas requires NaDSet to be a second order logic with both first and second order paramenters. However, it does not require two orders of bound variables.

In order to further the goal of a computational denotational semantics, a slightly simplified version of NaDSet is used in this paper. For those familiar with natural deduction presentations of first order logic, a full detailed description of NaDSet is unnecessary for an understanding of this paper; the following points cover the essentials:

- Abstraction terms take only the form {**v** | **F** } in the simplified NaDSet, where **v** is a variable and **F** is any formula. The variable **v** is bound in the term.
- A term is first order if no second order parameter occurs in it; otherwise it is second order. Thus if 'C' is a second order constant, 'a' a first order constant, and 'P' a second order parameter, {u | u:C ∨ u=a} is first order, while {u | u:P ∨ v=a} is second order. Note that the variable v is free in the second of these terms.
- A term or formula in which no variable has a free occurrence is said to be closed.
- A closed formula **r**:**s** is atomic if **r** is first order, and **s** is a second order parameter or constant.
- An interpretation consists of a set $\mathbb{D}$ of subsets of **d** and a function Φ for which Φ[**p**] ∈ **d**, for each first order parameters **p**, and Φ[**PC**] ∈ $\mathbb{D}$, for each second order parameter or constant **PC**.
- An atomic formula **r**:**PC** is assigned the truth value 'true' if Φ[**r**] ∈ Φ[**PC**], otherwise it is assigned 'false'. Here Φ[**r**] is defined to be the term obtained from **r** by replacing each occurrence of a parameter **p** in **r** by Φ[**p**]; Φ[**r**] is necessarily a member of **d**.
- Truth values are assigned to nonatomic closed formulas by transfinite induction as was done in [Gilmore80,86]. Not all closed formulas receive a truth value. Further, it is possible for an interpretation to assign a formula both 'true' and'false' if $\mathbb{D}$ does not contain sufficiently many subsets of **d**. Using the terminology of [Henkin50], an inconsistent interpretation is a frame that is not a general model. Consistent interpretations of NaDSet are constructed in [Gilmore91a] in the course of proving completeness.

2.3. Logical Syntax

The logical syntax or proof theory used for NaDSet in this paper is the method of semantic trees derived from the semantic tableaux method of [Beth55]. Since the presentation of first order logic in [Smullyan68] is also derived from semantic tableaux, this book may be helpful to readers.

Beth's method is interpreted here as a way of constructing derivations for Gentzen sequents. Given

such a sequent

$\mathbf{F_1, ..., F_m \rightarrow G_1, ..., G_n}$

the method systematically tries to construct a counter-example for the sequent; that is, an interpretation in which the antecedent formulas $\mathbf{F_1, ..., F_m}$ of the sequent are all true and the succedent formulas $\mathbf{G_1, ..., G_n}$ all false. The assertion of the truth or falsehood of a formula is expressed in a semantic tree by prefixing the formula respectively with a '+' or '-' sign, instead of with 'T' and 'F' in Smullyan's adaptation. The method begins with a tree with a single branch the nodes of which consist of the formulas $\mathbf{F_1, ..., F_m}$ prefixed with '+' for 'true', and the formulas $\mathbf{G_1, ..., G_n}$ prefixed with '-' for 'false'. The tree is extended by adding a signed formula as a new node, or by branching with two signed formulas as new nodes. Details need not be provided here since the proof theory will be richly illustrated below with examples of derivations needed for the main result of this paper; see in particular §5.1.1. It is crucial to note, however, that the closure condition on a branch is stricter than that of Beth and Smullyan: Only an atomic formula appearing prefixed with '+' and with '-' can close a branch. The tree is closed if every branch is closed. A closed semantic tree is a derivation for the sequent that initiated it.

2.4. Definitions

Definitions take the following form:

$\mathbf{r{=}s}$ **for** $\forall z\ (\mathbf{r{:}z \supset s{:}z})$

where **r** and **s** are any terms in which **z** is not a free variable. The definition can be regarded as stating an abbreviation: The formula on the left can be understood to be an abbreviation of the formula on the right. This definition of '=' can be shown to have all the properties of an intensional identity [Gilmore89] expressed here as sequents:

$\mathbf{r{=}s \rightarrow r{=}s}$

$\mathbf{\rightarrow r{=}r}$

$\mathbf{r{=}s \rightarrow s{=}r}$

$\mathbf{F, r{=}s \rightarrow [s/r]F}$

where **r** and **s** are any closed first order terms, **F** any formula, and **[s/r]** a substitution operator that replaces an occurrence of **r** in **F** by **s**. The first of the sequences, the law of excluded middle for the identity of first order terms, is peculiar to NaDSet. Unlike first order logic, $\mathbf{F \rightarrow F}$ is not necessarily a derivable sequent for a closed formula **F**; instances require derivations. When the sequent $\mathbf{F \rightarrow F}$ has been derived, + **F** and - **F** appearing on the same branch of a derivation is sufficient to close the branch, just as in first order logic.

NaDSet is an intensional logic and the usual extensionality axiom is inconsistent with it. That is, extensional identity defined as

$\mathbf{r{=}_e s}$ **for** $[\forall u{:}\mathbf{r}]\ u{:}\mathbf{s} \wedge [\forall u{:}\mathbf{s}]\ u{:}\mathbf{r}$

must be distintinguished from r=s. Although

$r=s \rightarrow r=_e s$

is derivable in NaDSet, its converse

$r=_e s \rightarrow r=s$

is inconsistent with NaDSet [Gilmore89].

Definitions of terms, such as the following definition of order pair, take the same form:

$\langle r,s\rangle$ **for** $\{u \mid r:C \wedge s:C\}$

where 'C' is a given second order constant, and r and s are any terms. This definition of ordered pair can also be shown to have all the expected properties [Gilmore89]. Expressed as sequents, these are:

$\langle r,s\rangle=\langle r',s'\rangle \rightarrow r=r' \wedge s=s'$

$r=r', s=s' \rightarrow \langle r,s\rangle=\langle r',s'\rangle$

Finally, given any terms $r_1, \ldots, r_n$, the finite set with just those terms as members is defined:

$\{r_1, \ldots, r_n\}$ **for** $\{u \mid u=r_1 \vee \ldots \vee u=r_n\}$

3. DOMAIN CONSTRUCTORS in NaDSet

The notation used for the four domain constructors in NaDSet differs slightly from that used in [Gordon79]:

Disjoint Union:	[D+R]
Cartesian Product:	[D×R]
Finite Power Set:	FP[D]
Finite Functions:	[D⇒R]

Here the notation FP[D] is used in place of D*; they are however the same set, namely the set of all finite sequences of elements from D. FP is called the finite power set constructor because finite sequences of elements from D can be interpreted as finite subsets, and are so interpreted when FP is used in the definition of [D⇒R]. The double arrow ⇒ has replaced the single arrow in [D→R] because the single arrow is reserved for Gentzen sequents.

The definition of FP[D] makes use of an end-of-sequence marker ⊥ in the usual fashion of lists. One assumption is necessary for ⊥:

⊥) $\langle r,s\rangle=\perp \rightarrow$

where r and s are any terms. The assumed derivability of these sequents in NaDSet ensures that ⊥ is a first order constant without any structure that can be expressed in terms of ordered pair.

It is not difficult to give a definition for each of the four constructors in NaDSet. But it is necessary to ensure that the definitions are composable to any depth. For example, it is necessary to ensure that constructed domains such as [[Id⇒Bv]⇒[Id⇒Bv]] have the properties expected of domains and can themselves be arguments for the constructors. For this reason, the following sequent is assumed to be derivable for any primitive domain **D**:

g) $p{:}\mathbf{D} \rightarrow p{:}\mathbf{D}$

where p is a first order parameter. This sequent expresses that p:**D** must be "grounded"; that is, a branch of a semantic tree is closed if both + p:**D** and - p:**D** occur on it. From (g) it is possible to conclude that $\mathbf{r}{:}\mathbf{D} \rightarrow \mathbf{r}{:}\mathbf{D}$ is derivable for any closed first order term **r** so that a branch of a semantic tree is closed if both + **r**:**D** and - **r**:**D** occur on it. (g) is an obvious assumption to make for the primitive domains; it must be derived for each constructor expression, assuming that the arguments of the constructor satisfy it.

3.1. Equivalence Relation on a Range

[**D**⇒**R**] will be defined to be the set of finite functions with domain **D** and range **R**. Before [**D**⇒**R**] can be so defined, it is necessary that there be an equivalence relation ≡[**R**] defined on **R** expressing the identity of values for the functions; that is, the following sequents must be derivable:

Rg) $\mathbf{r}{:}\mathbf{R}, \mathbf{s}{:}\mathbf{R}, \mathbf{r} \equiv[\mathbf{R}]\ \mathbf{s} \rightarrow \mathbf{r} \equiv[\mathbf{R}]\ \mathbf{s}$ (**r** and **s** first order closed terms)

R1) $\rightarrow [\forall x{:}\mathbf{R}]\ x \equiv[\mathbf{R}]\ x$

R2) $\rightarrow [\forall x,y{:}\mathbf{R}]\ (x \equiv[\mathbf{R}]\ y \supset y \equiv[\mathbf{R}]\ x)$

R3) $\rightarrow [\forall x,y,z{:}\mathbf{R}]\ (x \equiv[\mathbf{R}]\ y \wedge y \equiv[\mathbf{R}]\ z \supset x \equiv[\mathbf{R}]\ z)$

where the usual infix notation has been used:

$\mathbf{r} \equiv[\mathbf{R}]\ \mathbf{s}$ **for** $\langle \mathbf{r},\mathbf{s} \rangle{:}\equiv[\mathbf{R}]$

The relation ≡[**R**] may be simply = if **R** is a primitive domain or a defined finite domain; or it could be $=_e$; or as will be seen in §3.6 it could be $=_\epsilon$. But before a constructed domain can serve as **R** in [**D** ⇒ **R**], an equivalence relation must be defined for it. Thus for each of the constructors, an equivalence relation must be defined assuming that the arguments have equivalence relations defined for them.

3.2. Definition of +

The disjoint union of two domains **D** and **R** is defined:

$[\mathbf{D}+\mathbf{R}]$ **for** $\{x \mid (x{:}\mathbf{D} \vee x{:}\mathbf{R}) \wedge \forall x\ (x{:}\mathbf{D} \wedge x{:}\mathbf{R} \supset x{=}\perp)\}$

The disjoint union is normally applied when the sets **D** and **R** are disjoint, but the end of sequence marker ⊥ may be common to two domains that are otherwise disjoint. For this reason, the following sequent must be assumed to be derivable before the disjoint union can be meaningfully applied:

+) $\rightarrow \forall x\ (x{:}D \wedge x{:}R \supset x{=}\perp)$

Assuming (g) for **D**, **D'**, **R** and **R'** the following sequents are derivable:

+g) $p{:}D{+}R \rightarrow p{:}D{+}R$

+m) $\forall x\ (x{:}D' \wedge x{:}R' \supset x{=}\perp),\ [\forall x{:}D]\ x{:}D',\ [\forall x{:}R]\ x{:}R' \rightarrow [\forall x{:}[D{+}R]]\ x{:}[D'{+}R']$

+e) $\forall x\ (x{:}D' \wedge x{:}R' \supset x{=}\perp),\ D{=_e}D',\ R{=_e}R' \rightarrow [D{+}R]{=_e}[D'{+}R']$

The sequent (+m) expresses that + is monotone in each of its arguments.

A relation $\equiv[D{+}R]$ on $[D{+}R]$ can be defined:

$r \equiv[D{+}R]\ s$ **for** $[\exists x,y{:}D]\ (x{=}r \wedge y{=}s \wedge x \equiv[D]\ y) \vee [\exists x,y{:}R]\ (x{=}r \wedge y{=}s \wedge x \equiv[R]\ y)$

The sequents corresponding to (Rg), (R1), (R2), (R3) can be derived, assuming that they can be derived for $\equiv[D]$ and $\equiv[R]$.

3.3. Definition of ×

The Cartesian product of two domains **D** and **R** is defined:

$[D{\times}R]$ **for** $\{w \mid [\exists x{:}D][\exists y{:}R]\ w{=}\langle x.y\rangle\}$

Assuming (g) for **D**, **D'**, **R** and **R'** the following sequents are derivable:

×g) $\rightarrow p{:}D{\times}R \rightarrow p{:}D{\times}R$

×1) $\langle r,s\rangle{:}[D{\times}R] \rightarrow r{:}D \wedge s{:}R$ (**r** and **s** first order)

×m) $[\forall x{:}D]\ x{:}D',\ [\forall y{:}R]\ y{:}R' \rightarrow [\forall w{:}[D{\times}R]\ w{:}[D{\times}R']$

×e) $\forall x\ (x{:}D' \wedge x{:}R' \supset x{=}\perp),\ D{=_e}D',\ R{=_e}R' \rightarrow [D{\times}R]{=_e}[D'{\times}R']$

A relation $\equiv[D{\times}R]$ on $[D{\times}R]$ can be defined:

$\langle r,s\rangle \equiv[D{\times}R] \langle r',s'\rangle$ **for** $r \equiv[D]\ r' \wedge s \equiv[R]\ s'$

The sequents corresponding to (Rg), (R1), (R2), (R3) can be derived, assuming that they can be derived for $\equiv[D]$ and $\equiv[R]$.

3.4. Definition of FP[D]

It is in this definition that the first use of ⊥ is made:

$FP[D]$ **for** $\{w \mid \forall z\ (\perp{:}z \wedge [\forall x{:}D][\forall v{:}z]\ \langle v,x\rangle{:}z \supset w{:}z)\}$

Note that new elements are appended on the right, and not on the left. So for example, the members of FP[{0}] are ⊥, <⊥,0>, <<⊥,0>,0>, ... Thus the last element of a sequent is appended last. This is useful for a later definition.

The members of FP[D] can be regarded as representatives for the finite subsets of **D**, with ⊥ representing the empty subset. The representation is not unique since an object may appear in a

sequence more than once, as the example of the previous paragraph illustrates. This causes no difficulties once the membership relation $\in$ and extensional identity $=_\in$ with respect to $\in$ have been defined; see §3.5 and §3.6 below.

The definition of FP[**D**] is a typical least set definition in NaDSet; FP[**D**] is the smallest set z for which

$$\bot:z \wedge [\forall x:\mathbf{D}][\forall v:z]\ \langle v,x\rangle:z$$

is true. The following sequents are all derivable, assuming (g) for **D** and **D'** and with FP[**D**] abbreviated to FP:

FPg) $p:FP \to p:FP$

FP1) $\to \bot:FP$

FP2) $\to [\forall x:\mathbf{D}]\ [\forall v:FP]\ \langle v,x\rangle:FP$

FPI) FP Induction for St, where $p:St \to p:St$

$[\forall x:\mathbf{D}]\ \bot:St,\ [\forall x:\mathbf{D}][\forall v:St]\ \langle v,x\rangle:St \to [\forall w:FP]\ w:St$

FP3) $\to [\forall w:FP](\ w=\bot \vee [\exists x:\mathbf{D}][\exists v:FP]\ w=\langle v,x\rangle)$

FPm) $[\forall x:\mathbf{D}]\ x:\mathbf{D}' \to [\forall x:FP[\mathbf{D}]]\ x:FP[\mathbf{D}']$

FPe) $\mathbf{D}=_e\mathbf{D}' \to FP[\mathbf{D}]=_eFP[\mathbf{D}']$

The derivability of (FPI) illustrates a point made in the introduction: This induction principle is derived, not asserted as a property of FP.

3.5. Definition of $\in[\mathbf{D}]$

The membership relation between members of **D** and members of FP[**D**] is also defined by a least set definition:

$$\in[\mathbf{D}]\ \textbf{ for } \{w \mid \forall z\ (\ [\forall x:\mathbf{D}][\forall v:FP[\mathbf{D}]]\ \langle x,\langle v,x\rangle\rangle:z \wedge [\forall x,y:\mathbf{D}][\forall v:FP[\mathbf{D}]]\ (\ \langle x,v\rangle:z \supset \langle x,\langle v,y\rangle\rangle:z) \supset w:z\ )\}$$

The customary infix notation for $\in$ will also be used when there is no risk of confusion in dropping [**D**] from $\in[\mathbf{D}]$:

$$\mathbf{r} \in \mathbf{s} \textbf{ for } \langle \mathbf{r},\mathbf{s}\rangle:\in[\mathbf{D}]$$

Assuming (g) for **D**, and dropping [**D**] from FP and $\in$, the following sequents are derivable:

$\in$g) $\mathbf{r}:FP,\ \mathbf{s}:FP,\ \langle \mathbf{r},\mathbf{s}\rangle:\in \to \langle \mathbf{r},\mathbf{s}\rangle:\in$ (**r** and **s** closed first order)

$\in$1) $\to [\forall x:\mathbf{D}][\forall v:FP]\ x \in \langle v,x\rangle$

$\in$2) $\to [\forall x,y:\mathbf{D}][\forall v:FP]\ (x \in v \supset x \in \langle v,y\rangle)$

$\in$3) $\to [\forall x:\mathbf{D}]\ \neg x \in \bot$

$\in$I) $\in$ Induction for St, where $p:St \to p:St$ is assumed derivable:

$[\forall x:\mathbf{D}][\forall v:FP]\ \langle x,\langle v,x\rangle\rangle:St,\ [\forall x,y:\mathbf{D}][\forall v:FP](\langle v,x\rangle:St \supset \langle x,\langle v,y\rangle\rangle:St) \to [\forall w:\in]\ w:St$

$\in$4) $\rightarrow \forall x \forall w\ (x \in w \supset x{:}\mathbf{D} \wedge w{:}FP)$

$\in$5) $\rightarrow [\forall w{:}\in][\exists x,y{:}\mathbf{D}][\exists v{:}FP]w=<x,<v,y>>)$

3.6. Definition of $=_\in[\mathbf{D}]$

The extensional identity of two finite subsets can be defined with respect to $\in$ in the usual way:

$<\mathbf{r},\mathbf{s}>:=_\in[\mathbf{D}]$ **for** $[\forall x{:}\mathbf{D}]\ (x \in \mathbf{r} \equiv x \in \mathbf{s})$

Note that $=_\in$ must be distinguished from $=_e$.

Assuming (g) for **D**, and dropping [**D**] from FP and $=_\in$, the following sequents are derivable:

$=_\in$g) $\mathbf{r}{:}FP, \mathbf{s}{:}FP, \mathbf{r} =_\in \mathbf{s} \rightarrow \mathbf{r} =_\in \mathbf{s}$ (**r** and **s** closed first order)

$=_\in$1) $\rightarrow [\forall x{:}FP]\ x =_\in x$

$=_\in$2) $\rightarrow [\forall x,y{:}FP]\ (x =_\in y \supset y =_\in x)$

$=_\in$3) $\rightarrow [\forall x,y,z{:}FP]\ (x =_\in y \wedge y =_\in z \supset x =_\in z)$

The four sequents express that $=_\in[\mathbf{D}]$ is an equivalence relation on FP[**D**], so that it may serve as the equivalence relation $\equiv$[FP[**D**]].

3.7. Definition of [D⇒R]

It is assumed that an equalence relation $\equiv$[**R**] is defined on **R**.

A preliminary definition is useful for the definition of [**D**⇒**R**]:

Fct[**D**,**R**,$\equiv$] **for** $\{w \mid [\forall x{:}\mathbf{D}][\forall y,y'{:}\mathbf{R}]\ (<x,y> \in w \wedge <x,y'> \in w \supset y \equiv[\mathbf{R}]\ y')\}$

Here $\in$[**D**×**R**] has been abbreviated to $\in$.

The set of finite functions with domain **D** and range **R** can now be defined.

[**D**⇒**R**] **for** $\{w \mid w{:}FP[\mathbf{D}\times\mathbf{R}] \wedge w{:}Fct[\mathbf{D},\mathbf{R},\equiv]\}$

The fact that w:Fct[**D**,**R**,$\equiv$] is a simple "add-on" restricting the members of [**D**⇒**R**] to being single valued functions, suggests that a semantics for non-determinism could be developed by simply dropping it and using FP[**D**×**R**] in place of [**D**⇒**R**]. However, this possibility is not explored in this paper.

Assuming (g) for **D**, **D'**, **R** and **R'**, and abbreviating $\in$[**D**×**R**] to $\in$, the following sequents are derivable:

FFg) $\mathbf{r}{:}[\mathbf{D}\Rightarrow\mathbf{R}] \rightarrow \mathbf{r}{:}[\mathbf{D}\Rightarrow\mathbf{R}]$

FF1) $\rightarrow \forall x,y\ [\forall w{:}[\mathbf{D}\Rightarrow\mathbf{R}]]\ (<x,y> \in w \supset x{:}\mathbf{D} \wedge y{:}\mathbf{R})$

FFm) $[\forall x{:}\mathbf{D}]\ x{:}\mathbf{D}', [\forall x{:}\mathbf{R}]\ x{:}\mathbf{R}' \rightarrow [\forall w{:}[\mathbf{D}\Rightarrow\mathbf{R}]\ w{:}[\mathbf{D}'\Rightarrow\mathbf{R}']$

FFe) $\mathbf{D}=_e\mathbf{D}', \mathbf{R}=_e\mathbf{R}' \rightarrow [\mathbf{D}\Rightarrow\mathbf{R}]=_e[\mathbf{D}'\Rightarrow\mathbf{R}']$

Since the members of $[\mathbf{D}\Rightarrow\mathbf{R}]$ are members of $FP[\mathbf{D}\times\mathbf{R}]$, an equivalence relation $=_\epsilon [\mathbf{D}\times\mathbf{R}]$ has already been defined on $[\mathbf{D}\Rightarrow\mathbf{R}]$.

4. DOMAIN EQUATIONS

4.1. An Example

A simple example adapted from [Gordon79] will be used to illustrate how solutions for domain equations can be defined within NaDSet. The primitive domains for this example are Bv and Id, while the complex domains State, Value, and Proc (Procedure) are defined by the following domain equations stated as usual in terms of intensional identity =:

State = [Id⇒Value]
Value = [Bv+ Proc]
Proc = [State⇒State]

One simplifying definition is useful prior to the definitions of State, Value, and Proc:

Cond[S,V,P] **for** $[\forall u:[Id\Rightarrow V]]\ u:S \wedge [\forall v:[Bv + P]]\ v:V \wedge [\forall w:[S\Rightarrow S]]\ w:P$

Here in a slight departure from previously used syntax, the upper case letters S, V, and P are used as variables. Note that each of the conjuncts of Cond[S,V,P] is obtained directly from one of the equations. The following definitions of State, Value, and Proc ensure that they are the minimum sets satisfying Cond[S,V,P]:

State **for** $\{u \mid \forall S,V,P\ (Cond[S,V,P] \supset u:S)\}$
Value **for** $\{v \mid \forall S,V,P\ (Cond[S,V,P] \supset v:V)\}$
Proc **for** $\{w \mid \forall S,V,P\ (Cond[S,V,P] \supset w:P)\}$

With these definitions the following three sequents are derivable:

EState) $\rightarrow State =_e [Id\Rightarrow Value]$

EValue) $\rightarrow Value =_e [Bv + Proc]$

EProc) $\rightarrow Proc =_e [State\Rightarrow State]$

Thus = of the domain equations has been replaced by $=_e$. The derivations of these sequents will make use of the following derivable sequents:

IS) $\rightarrow \forall S,V,P\ (Cond[S,V,P] \supset [\forall u:State]\ u:S)$ State Induction

IV) $\rightarrow \forall S,V,P\ (Cond[S,V,P] \supset [\forall v:Value]\ v:V)$ Value Induction

IP) $\rightarrow \forall S,V,P\ (Cond[S,V,P] \supset [\forall w:Proc]\ w:P)$ Procedure Induction

C) $\rightarrow Cond[State, Value, Proc]$

4.2. Systems of Domain Equations

Consider now a system of domain equations as defined in the introduction:

$\mathbf{D_1} = \mathbf{T_1}\ [\mathbf{D_1}, \ldots, \mathbf{D_n}]$

$\mathbf{D_2} = \mathbf{T_2}\ [\mathbf{D_1}, \ldots, \mathbf{D_n}]$

...

$\mathbf{D_n} = \mathbf{T_n}\ [\mathbf{D_1}, \ldots, \mathbf{D_n}]$

where each $\mathbf{T_i}\ [\mathbf{D_1}, \ldots, \mathbf{D_n}]$ is an expression constructed out of domain variables $\mathbf{D_1}, \ldots, \mathbf{D_n}$ and finite and primitive domains using the domain constructors. A system of domain equations is said to be reduced if each $\mathbf{T_i}\ [\mathbf{D_1}, \ldots, \mathbf{D_n}]$ takes one of the following four forms:

$[\mathbf{D}+\mathbf{R}]$, $[\mathbf{D}\times\mathbf{R}]$, $\mathrm{FP}[\mathbf{D}]$, or $[\mathbf{D}\Rightarrow\mathbf{R}]$

where each **D** and **R** is a primitive domain, a defined finite domain, or a domain variable $\mathbf{D_i}$ for some i. The example system of domain equations in §4.1 is a reduced system.

The following definitions are suitable for any system of equations, reduced or not:

$\mathrm{Cond_i}[S_1, \ldots, S_n]$ **for** $[\forall u{:}T_i[\ S_1, \ldots, S_n]]\ u{:}S_i$ $\qquad n \geq i \geq 1$

$\mathrm{Cond}[S_1, \ldots, S_n]$ **for** $\mathrm{Cond_1}[S_1, \ldots, S_n] \wedge \ldots \wedge \mathrm{Cond_n}[S_1, \ldots, S_n]$

$\mathbf{D_i}$ **for** $\{v \mid \forall S_1, \ldots, S_n\ (\ \mathrm{Cond}[S_1, \ldots, S_n] \supset v{:}S_i\ )\ \}$ $\qquad n \geq i \geq 1$

The following induction principles are derivable in the same manner as (IS), (IV), and (IP):

ID$_i$) $\qquad \rightarrow \forall S_1, \ldots, S_n\ (\ \mathrm{Cond}[S_1, \ldots, S_n] \supset [\forall v{:}\mathbf{D_i}]\ v{:}S_i\ )$ $\qquad n \geq i \geq 1$

A reduced system of domain equations can be obtained from any system by introducing new domain variables $\mathbf{D_i}$ to reduce each $\mathbf{T_i}\ [\mathbf{D_1}, \ldots, \mathbf{D_n}]$ to one of the four simplified forms. Because of the derivability of the sequents (+e), (×e), (FPe), and (FFe), a solution to the reduced system provides a solution to the system itself. For this reason, derivations will be provided for the following sequents assuming the system of equations is reduced:

C) $\qquad \rightarrow \mathrm{Cond}[\mathbf{D_1}, \ldots, \mathbf{D_n}]$

Ei) $\qquad \rightarrow \mathbf{D_i} =_e \mathbf{T_i}\ [\mathbf{D_1}, \ldots, \mathbf{D_n}]$ $\qquad n \geq i \geq 1$

5. DERIVATIONS

This section has two parts §5.1 and §5.2. In §5.1, selected sequents from among those claimed to be derivable in §3 will be provided with derivations. The purpose of the first part is to introduce the semantic tree proof theory of NaDSet. Sequents have been selected whose derivations illustrate

particular features worth highlighting. In §5.2, abbreviated derivations are given for sequents claimed to be derivable in §4.2, assuming that all the sequents claimed to be derivable in §3 have been given derivations. Derivations of sequents not given in §5.1 or §5.2, will be left as exercises for the reader.

5.1. Selected Derivations

5.1.1 Derivation of (+g)

The role of the assumption (g) in derivations is illustrated with this derivation. The sequent to be derived is $p:\mathbf{D}+\mathbf{R} \rightarrow p:\mathbf{D}+\mathbf{R}$, assuming that $p:\mathbf{D} \rightarrow p:\mathbf{D}$ and $p:\mathbf{R} \rightarrow p:\mathbf{R}$ are both derivable. As will be explained in 5.1.2, the latter means that both + p:**D** and - p:**D**, or both + p:**R** and - p:**R**, on the same branch closes it, even though p:**D** and p:**R** are not necessarily atomic. This derivation also relies upon an instance of the first sequent for =, namely $p=\perp \rightarrow p=\perp$.

Unlike later derivations, this derivation will be given in full and annotated to provide details of the method of semantic trees. The two initial assumptions come from the sequent to be derived as described in §2.3.

+ p:**D**+**R**	1. Initial assumption
– p:**D**+**R**	2. Initial assumption
+ p:{x | (x:**D** ∨ x:**R**) ∧ ∀x (x:**D** ∧ x:**R** ⊃ x=⊥)}	3. Definition of +
+ (p:**D** ∨ p:**R**) ∧ ∀x (x:**D** ∧ x:**R** ⊃ x=⊥)	4. (1) by meaning of { }
+ (p:**D** ∨ p:**R**)	5. (4) by meaning of ∧
+ ∀x (x:**D** ∧ x:**R** ⊃ x=⊥)	6. (4) by meaning of ∧
- p:{x | (x:**D** ∨ x:**R**) ∧ ∀x (x:**D** ∧ x:**R** ⊃ x=⊥)}	7. Definition of +
- (p:**D** ∨ p:**R**) ∧ ∀x (x:**D** ∧ x:**R** ⊃ x=⊥)	8. (2) by meaning of { }
- (p:**D** ∨ p:**R**) -∀x (x:**D** ∧ x:**R** ⊃ x=⊥)	9. (8) by meaning of ∧ branch splits
(i) (ii)	
- (p:**D** ∨ p:**R**)	10. Repeat of one possibility from (9)
- p:**D**	11. (10) by meaning of ∨
- p:**R**	12. (10) by meaning of ∨
+ p:**D** + p:**R**	13. (5) by meaning of ∨ branch splits
==== ====	14. Branches close because of (11), (12), (13) and assumption (g) for **D** and **R**
(ii)	15. Repeat second possibility from (9)
-∀x (x:**D** ∧ x:**R** ⊃ x=⊥)	16. (15) New parameter p because of -∀
- (p:**D** ∧ p:**R** ⊃ p=⊥)	17. (16) by meaning of ⊃
+ p:**D** ∧ p:**R**	

$-\ p{=}\perp$	18. (16) by meaning of $\supset$
$+\ p{:}\mathbf{D}$	19. (17) by meaning of $\wedge$
$+\ p{:}\mathbf{R}$	20. (17) by meaning of $\wedge$
$+\ (p{:}\mathbf{D} \wedge p{:}\mathbf{R} \supset p{=}\perp)$	21. (6) by meaning of $\forall$
$-\ p{:}\mathbf{D} \wedge p{:}\mathbf{R}$ $\quad +\ p{=}\perp$	22. (21) by meaning of $\supset$ branch splits
$-\ p{:}\mathbf{D}$ $\quad -\ p{:}\mathbf{R}$	23. Right branch closes (18), (22)
	24. (22) by meaning of $\wedge$ branch splits
	25. Both branches close because of (19), (20), (24) and assumption (g) for **D** and **R**

Henceforth, derivations will not be presented in the detail provided here. For example, lines (3) and (7) can be omitted with lines (4) and (8) known to be consequences of the definition of +. Further, lines of the derivations will not be numbered, so that no backward referencing will be given.

The next two subsections reference this derivation and other derivations appearing below to illustrate aspects of semantic tree derivations.

5.1.2. Cut: Using Previously Derived Sequents

The derivation in 5.1.1 makes use of the previously derived sequents $p{:}\mathbf{D} \rightarrow p{:}\mathbf{D}$, $p{:}\mathbf{R} \rightarrow p{:}\mathbf{R}$, and $p{=}\perp \rightarrow p{=}\perp$ to close branches of the derivation. For example, at line (14) two branches close because of the first two sequents, as is the case also at line (25). The third sequent is used at line (23). This is one of the simplest uses of previously derived sequents. Since the sequent is known to have a semantic tree derivation, any branch of a derivation on which both $+p{=}\perp$ and $-p{=}\perp$ appear must necessarily close, since the tree derivation for the sequent can be appended to the branch.

This technique generalizes for any given derivable sequent $\mathbf{F}_1, ..., \mathbf{F}_m \rightarrow \mathbf{G}_1, ..., \mathbf{G}_n$. Consider a branch on which each of $\mathbf{F}_1, ..., \mathbf{F}_m$ appears prefixed with '+', and each of $\mathbf{G}_1, ..., \mathbf{G}_n$ appears prefixed with '-'. Necessarily the branch can be closed since it can be extended to the full derivation of the sequent.

What happens if say $\mathbf{G}_1$ does not appear on the branch prefixed with a '-', but all else is the same? To simplify the discussion, assume that the derived sequent is $\mathbf{F} \rightarrow \mathbf{G}$, and that $+\mathbf{F}$ appears on the branch. Then by using a rule of semantic trees corresponding to the cut rule of the Gentzen calculus. $+\mathbf{G}$ can be added to the branch. This is illustrated here:

...

+F

+G -G

===

The split in the branch is introduced by the cut rule, with +G and -G introduced into the two branches created. The right branch necessarily closes because there is a closed tree derivation with +F and -G as initial assumptions. Thus modus ponens is justified by the cut rule. Examples of this use of cut appear in the derivation of (FP3) in section 5.1.4 below on the lines marked (=) which make use of the last derivable sequent for = listed in §2.

A variation of this application of cut allows the addition of +G to any branch, provided $\rightarrow$ G is derivable. Examples of this use of cut appear in the derivation of (FP3) in section 5.1.4 below on the lines marked (FP1) and (FP2).

5.1.3. Decisions, Decisions

In line (16) of the derivation in 5.1.1, a new first order parameter p is introduced. In the fifth line of the derivation of (FFg) in 5.1.4 below, a new second order parameter P is introduced. The proof theory does not require that a second order parameter be introduced at that line; a first order parameter p could just as well have been introduced. However in that case the branches of the proposed derivation do not close since $\perp$:p is not an atomic formula when p is first order. The decision as to whether a newly introduced parameter should be first or second order, is one of the decisions that has to be made in the construction of a derivation. Generally it is not a difficult decision. At worst an attempt at a derivation can be made with a first order parameter, and a second attempt with a second order parameter.

A truly difficult and creative decision must be made when a closed term must be chosen to replace a variable z in signed formulas + $\forall$z (...z...) or - $\exists$z (...z...); this is true for first order logic derivations as well. Line (21) of the derivation in 5.1.1 is an example, although the choice of p to replace x was not a difficult one. The derivation of (FPm) in 5.1.4 provides a better example.

5.1.4. Derivations of Sequences for FP

A least set definition, such as that of FP, has immediate consequences that are well illustrated with the sequences for FP. At the same time they introduce the role of second order parameters, and make use of cut as described in 5.1.2.

Derivation of (FPg)

+ p:FP

- p:FP

+∀z (⊥:z ∧ [∀x:**D**][∀v:z] <v,x>:z ⊃ p:z)

-∀z (⊥:z ∧ [∀x:**D**][∀v:z] <v,x>:z ⊃ p:z)

- (⊥:P ∧ [∀x:**D**][∀v:P] <v,x>:P ⊃ p:P)

+ (⊥:P ∧ [∀x:**D**][∀v:P] <v,x>:P ⊃ p:P)

+ ⊥:P

+ [∀x:**D**][∀v:P] <v,x>:P ⊃ p:P

- ⊥:P — - [∀x:**D**][∀v:P] <v,x>:P ⊃ p:P

==== ==========

The closing of the first branch is immediate because ⊥:P is an atomic formula. The closing of the second branch is not immediate; however, since a derivation for the sequent

[∀x:**D**][∀v:P] <v,x>:P ⊃ p:P → [∀x:**D**][∀v:P] <v,x>:P ⊃ p:P

can be quite simply constructed, the completion of the derivation is left for the reader.

Since FP is supposed to be the least set z satisfying ⊥:z ∧ [∀x:**D**][∀v:z] <v,x>:z, it is not surprising that it satisfies this formula as shown in the next two derivations.

Derivation of (FP1)

- ⊥:FP

-∀z (⊥:z ∧ [∀x:**D**][∀v:z] <v,x>:z ⊃ ⊥:z)

- (⊥:P ∧ [∀x:**D**][∀v:P] <v,x>:P ⊃ ⊥:P)

+ ⊥:P ∧ [∀x:**D**][∀v:P] <v,x>:P

- ⊥:P

+ ⊥:P

=====

Derivation of (FP2)

- [∀x:**D**][∀v:FP] <v,x>:FP

+ r:FP

+ p:**D**

- <r,p>:FP

-∀z (⊥:z ∧ [∀x:**D**][∀v:z] <v,x>:z ⊃ <r,p>:z)

-(⊥:P ∧ [∀x:**D**][∀v:z] <v,x>:P ⊃ <r,p>:P)

+ [∀x:**D**][∀v:P] <v,x>:P

- <r,p>:P

+ <r,p>:P

=======

Derivation of (FPI)

+ ⊥:St

+ [∀v:St][∀x:**D**] <v,x>:St

- [∀w:FP] w:St

+ r:FP

- r:St

+∀z (⊥:z ∧ [∀x:**D**][∀v:z] <v,x>:z ⊃ r:z)

+ (⊥:St ∧ [∀x:**D**][∀v:St] <v,x>:St ⊃ r:St)

+ r:St

==== .

Derivation of (FP3)

This provides an example of an application of FP induction, with St taken to be {w | (w=⊥ ∨ [∃x:**D**][∃v:FP] w=<v,x>)}.

Need to prove

a) → ⊥:St

b) → [∀v:St][∀x:**D**] <v,x>:St

Derivation of (a)

- ⊥:St

- (⊥=⊥ ∨ [∃x:**D**][∃v:FP] ⊥=<v,x>)

- ⊥=⊥

=====

Derivation of (b)

- [∀v:St][∀x:**D**] <v,x>:St

+ r:St

+ p:**D**

- <r,p>:St

- (<r,p>=⊥ ∨ [∃x:**D**][∃v:FP] <r,p>=<v,x>)

- [∃x:**D**][∃v:FP] <r,p>=<v,x>

+ (r=⊥ ∨ [∃x:**D**][∃v:FP] r=<v,x>)

+ r=⊥	+ [∃x:**D**][∃v:FP] r=<v,x>
(i)	(ii)

+ ⊥:FP (FP1)

+ r:FP (=)

- <r,p>=<r,p>

==========

(ii)

+ [∃x:**D**][∃v:FP] r=<v,x>

+ p':**D**

+ r':FP

+ r=<p',r'>

+ [∀x:**D**] [∀v:FP] <v,x>:FP (FP2)

+ <p',r'>:FP

+ r:FP (=)

- <r,p>=<r,p>

==========

Derivation of (FPm)

+∀x (x:**D** ⊃ x:**D'**)

-∀x (x:FP[**D**] ⊃ x:FP[**D'**])

+ r:FP[**D**]

- r:FP[**D'**]

-∀z (⊥:z ∧ [∀x:**D'**][∀v:z] <v,x>:z ⊃ r:z)

- (⊥:P ∧ [∀x:**D'**][∀v:P] <v,x>:P ⊃ r:P)

+ ⊥:P

+ [∀x:**D'**][∀v:P] <v,x>:P

- r:P

+∀z (⊥:z ∧ [∀x:**D**][∀v:z] <v,x>:z ⊃ r:z)

Here a choice must be made of a term to substitute for z. A substitution of P for z leads to no progress. A substitution of {w | w:P ∧ w≠r}, on the other hand, moves the derivation along because r:{w | w:P ∧ w≠r} is clearly false. Thus define St for {w | w:P ∧ w≠r}, and substitute St:

+ (⊥:St ∧ [∀x:**D**][∀v:St] <v,x>:St ⊃ r:St)

- ⊥:St ∧ [∀x:**D**][∀v:St] <v,x>:St + r:St

--- +r:P

- ⊥:St - [∀x:**D**][∀v:St] <v,x>:St ====

(i) (ii)

- ⊥:St

- ⊥:P ∧ ⊥≠r

- ⊥:P -⊥≠r

==== + ⊥=r

- ⊥:P

====

(ii)

- [∀x:D][∀v:St] <v,x>:St
+ p:D
+ r':St
- <r,p'>:St
+ r':P ∧ r'≠r
+ r':P
+ (p:D ⊃ p:D')
+ p:D'
+ <r,p>:P
- <r',p>:P ∧ <r,p>≠r
+ <r,p>=r
+ r:P

=====

5.2. Derivations of Sequents of §4.2

Derivation of ($\mathbb{D}_i$), i = 1, ... , n

-∀S_1, ... , S_n (Cond[S_1, ... , S_n] ⊃ [v:D_i] v:S_i)
+ Cond[P_1, ... , P_n]
+ p:D_i
- p:P_i
+∀S_1, ... , S_n (Cond[S_1, ... , S_n] ⊃ p:S_i)
+ (Cond[P_1, ... , P_n] ⊃ p:P_i)
- Cond[P_1, ... , P_n]

==========

Derivation of (C)

- Cond[D_1, ... , D_n]
- $Cond_1$[D_1, ... , D_n] ∧ ... ∧ $Cond_n$[D_1, ... , D_n]

---

(1) - $Cond_1$[D_1, ... , D_n]

...

(n) - $Cond_n$[D_1, ... , D_n]

It is sufficient to show that the branch including (1) closes; the other cases are similar.

- $Cond_1[D_1, \ldots, D_n]$

- $[\forall u{:}T_1[D_1, \ldots, D_n]]\ u{:}D_1$

+ $p{:}T_1[D_1, \ldots, D_n]$

- $p{:}D_1$

-$\forall S_1, \ldots, S_n\ (\ Cond[S_1, \ldots, S_n] \supset p{:}S_1)$

+ $Cond[P_1, \ldots, P_n]$

- $p{:}P_1$

+ $[\forall v{:}D_i]\ v{:}P_i$ (ID_i)

+ $p{:}T_1[P_1, \ldots, P_n]$ From + $p{:}T_1[D_1, \ldots, D_n]$ using (+m), (×m), (FPm), or (FFm) depending upon which of the four forms T_1 takes: [D+R], [D×R], FP[D], or [D⇒R].

+ $Cond_1[P_1, \ldots, P_n]$

+ $[\forall u{:}T_1[P_1, \ldots, P_n]]\ u{:}P_1$

+ $p{:}P_1$

======

Derivation of (E1) as an example for other (Ei), i = 2, ... , n
It is sufficient to provide derivations for

(a) → $[\forall x{:}T_1[D_1, \ldots, D_n]]\ x{:}D_1$

(b) → $[\forall x{:}D_1]\ x{:}T_1[D_1, \ldots, D_n]$

Derivation of (a)

- $[\forall x{:}T_1[D_1, \ldots, D_n]]\ x{:}D_1$

+ $Cond[D_1, \ldots, D_n]$ (C)

+ $Cond_1[D_1, \ldots, D_n]$

+ $[\forall u{:}T_1[D_1, \ldots, D_n]]\ u{:}D_1$

==========

Derivation of (b)

- $[\forall x{:}D_1]\ x{:}T_1[D_1, \ldots, D_n]$

+ $p{:}D_1$

- $p{:}T_1[D_1, \ldots, D_n]$

+ $Cond[D_1, D_2, \ldots, D_n]$ (C)

+$\forall S_1, \ldots, S_n\ (\ Cond[S_1, \ldots, S_n] \supset p{:}S_1)$

- $Cond[T_1, D_2, \ldots, D_n]$ where T_1 for $T_1[D_1, \ldots, D_n]$

---

(1) - $Cond_1[T_1, D_2, \ldots, D_n]$

...

(n) - $Cond_n[T_1, D_2, \ldots, D_n]$

For each i, $Cond_i[T_1, D_2, \ldots, D_n]$ is $[\forall u{:}T_i[T_1, D_2, \ldots, D_n]]\ u{:}D_i$, and $T_i[D_1, D_2, \ldots, D_n]$ is one of the terms [D+R], [D×R], FP[D], or [D⇒R], where each of D and R is a primitive domain, a finitely defined domain, or D_j, for some j. Therefore the following sequent is derivable from (+m), (×m), (FPm), or (FFm) depending upon which of the forms that T_i takes:

(im) $[\forall u{:}T_1[D_1, D_2, \ldots, D_n]]\ u{:}D_1 \rightarrow [\forall u{:}T_i[T_1, D_2, \ldots, D_n]]\ u{:}T_i[D_1, D_2, \ldots, D_n]$

Continue now with branch (i):

(i)

- $Cond_i[T_1, D_2, \ldots, D_n]$

+ $Cond_i[D_1, D_2, \ldots, D_n]$

+ $[\forall u{:}T_1[D_1, D_2, \ldots, D_n]]\ u{:}D_1$

+ $[\forall u{:}T_i[T_1, D_2, \ldots, D_n]]\ u{:}T_i[D_1, D_2, \ldots, D_n]$ (im)

+ $[\forall u{:}T_i[D_1, D_2, \ldots, D_n]]\ u{:}D_i$

+ $[\forall u{:}T_i[T_1, D_2, \ldots, D_n]]\ u{:}D_i$

+ $Cond_i[T_1, D_2, \ldots, D_n]$

==========

6. SEMANTICS OF PROGRAMMING LANGUAGES

In [Gilmore&Tsiknis91a] the semantics of a language used as an illustration in [Stoy77] was developed in NaDSet and a fully formal proof of the equivalence of two programs was given. The simple nature of the language permitted the definition of the semantics within NaDSet without the use of recursively defined domain equations. The formal semantics for the example language TINY developed in §2.4 of [Gordon79] can be similarly defined. However, what is called standard semantics in §5 of [Gordon79] using continuations cannot be developed in the same way. The demonstration that this standard semantics can be developed within NaDSet is provided in [Gilmore&Tsiknis92]. But a more fundamental question remains unanswered.

6.1. Scott Solutions vs NaDSet Solutions

The rather elementary character of the NaDSet solutions in contrast to the Scott solutions for domain equations naturally raises the question whether the NaDSet solutions are sufficient to support a denotational semantics. The difference between the two kinds of solutions to domain equations can

be illustrated with the single domain equation

1) $\mathbf{D}^{\omega} = [\mathbf{D}^{\omega} \times \mathbf{D}]$

obtained from the one appearing on page 34 of [Gordon79] by interchanging the arguments **D** and $\mathbf{D}^{\omega}$ of the Cartesian product, so as to match the definition of finite sequences given in §3.4. The Scott solution domain $\mathbf{D}^{\omega}$ defined by the equation consists of all infinite tuples $...<<<<\perp,d_1>,d_2>,d_3>$, ... of members d_1, d_2, d_3, ... of **D**. These infinite tuples are limit lists, as they are described in [Scott70]. The NaDSet solution domain to equation (1), on the other hand, is actually the empty set ∅, since

$\rightarrow \varnothing =_e [\varnothing \times \mathbf{D}]$

is derivable. But consider the NaDSet solution to the domain equation

2) $\mathbf{D}^{\omega 1} = [\mathbf{D} + [\mathbf{D}^{\omega 1} \times \mathbf{D}]]$

The set $\mathbf{D}^{\omega 1}$ so defined is extensionally identical with the union of all the sets **D**, [D×D], [[D×D]×D], [[[D×D]×D]×D], ... ; no limit list is a member. Indeed, since FP[D] differs from $\mathbf{D}^{\omega 1}$ only in representing the members of **D** as pairs $<\perp,d>$, FP[D] can serve in place of $\mathbf{D}^{\omega 1}$.

Not only are limit lists unnecessary for the solution of domain equations in NaDSet, their inclusion would lead to additional complications of a fundamental kind. Consider, for example, limit lists of the domain {0,1}, that is, all infinite sequences of 0's and 1's. Unless they are explicitly restricted to being computable, under some given definition of computable, the meaning of such lists and the properties they possess become dependant upon the set theory in which they are defined. This is a consequence of NaDSet's method of avoiding the paradoxes, in particular Cantor's paradox; details are provided in [Gilmore89].

6.2. Program Semantics without Limits

The difference between Scott solutions and NaDSet solutions is apparent in domain equations of an even simpler kind than (1). Consider the equation on page 15 of [Gordon79]

(ii) Memory = [Ide⇒[Value+{unbound}]]

Thus Memory is the domain of all functions from the domain Ide to the domain [Value+{unbound}], where by 'function' is meant a total single-valued continuous function. Thus the Scott solution to this domain equation is an infinite object.

Semantics for TINY can be developed in NaDSet using the following equation in place of (ii):

Memory = [Ide⇒Value],

provided the end-of-sequence marker ⊥ is included in Value. Thus Memory is the domain of all finite functions from the domain Ide to the domain Value. For a given **f** ∈ Memory and **i** ∈ Ide, **i** is unbound for **f** if **f** is not defined for **i**. Formally, this can be defined:

Unbound[**f**,**i**] **for** [∀v:Value] ~<**i**,v> ∈ **f**

Thus within NaDSet, the limit functions that are members of the domain Memory for the Scott solution to (ii), are replaced by the finite functions of which they are the limit.

But what about those uses made of the limit lists in non-terminating programs? For example, what about the TINY program

x:=0; while true do (output x; x:=x+1)

discussed on page 60 of [Gordon79] ? The output using Scott solutions is the infinite sequence **0.1.2.3 ...** , a limit list. The output using NaDSet solutions does not require the use of limits. Full details as to why not are deferred to [Gilmore&Tsiknis92] but the basic reason is the same: A limit list in a Scott solution is replaced by a sequences of finite lists of which the limit list is the limit.

6.3. Non-Well-Founded Sets

Systems of set equations different from systems of domain equations were introduced in [Axcel88] to characterize the non-well-founded sets of ZFA, which is ZF with the axiom of foundation replaced by an axiom of anti-foundation. An example of a system of one such equation is;

$\Omega = \{\Omega\}$

The solution of such an equation in NaDSet [Gilmore91c] has an entirely different character from a domain such as $\mathbf{D}^{\omega 1}$, the reason being that $\{\Omega\}$, unlike $[\mathbf{D}+[\mathbf{D}^{\omega 1}\times\mathbf{D}]]$, is not monotone.

7. REFERENCES

Aczel, Peter

(88) Non-Well-Founded Sets, CSLI Lecture Notes, No. 14, Stanford University

Beth, E.W.

(55) Semantic Entailment and Formal Derivability, *Mededelingen de Koninklijke Nederlandse Akademie der Wetenschappen, Afdeeling Letterkunde, Nieuwe Reeks,* 18, no.13, 309-342.

Gilmore, Paul C.

(80) Combining Unrestricted Abstraction with Universal Quantification, To H.B. Curry: Essays on Combinatorial Logic, Lambda Calculus and Formalism, Editors J.P. Seldin, J.R. Hindley, Academic Press, 99-123.

(86) Natural Deduction Based Set Theories: A New Resolution of the Old Paradoxes, *Journal of Symbolic Logic*, 51, 393-411.

(89) How Many Real Numbers are There?, UBC Computer Science Department Technical Report 89-7. Revised October, 1991. To appear in *The Annals of Pure and Applied Logic.*

(91a) The Consistency and Completeness of an Extended NaDSet, UBC Computer Science Department Technical Report 91-17. To appear in *The Annals of Pure and Applied Logic.*

(91b) Logic, Sets, and Mathematics, to appear in *Mathematical Intelligencer*.
(91c) Non-Well-Founded Sets in NaDSet, UBC Computer Science Department Technical Report 91-32.

Gilmore, Paul C. & Tsiknis, George K.
(91a) Logical Foundations for Programming Semantics, a paper presented to the Sixth Workshop on the Mathematical Foundations of Programming Semantics, Kingston, Ontario, Canada, May 15-19. To appear in *Theoretical Computer Science*.
(91b) A Formalization of Category Theory in NaDSet, a paper presented to the Sixth Workshop on the Mathematical Foundations of Programming Semantics, Kingston, Ontario, Canada, May 15-19. To appear in *Theoretical Computer Science*.
(91c) Formalizations of an Extended NaDSet, UBC Computer Science Department Technical Report 91-15, August, 1991.
(92) Continuation Semantics in NaDSet, In Preparation.

Henkin, Leon
(50) Completeness in the Theory of Types, *J. Symbolic Logic*, 15, 81-91.

Gordon, Michael J.C.
[79] The Denotational Description of Programming Languages, Springer-Verlag

Scott, Dana
(70) Outline of a Mathematical Theory of Computation, Technical Monograph PRG-2, Oxford University Computing Laboratory.

Smullyan, Raymond
(68) First Order Logic, Springer-Verlag

Stoy, Joseph E.
(77) Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory, MIT Press

Tsiknis, George K.
(91) Applications of a Natural Deduction Based Set Theory, UBC Department of Computer Science PhD thesis.