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

LOGICAL FOUNDATIONS
for
PROGRAMMING SEMANTICS

Paul C. Gilmore* and George K. Tsiknis

Technical Report TR 90-22

August, 1990

ABSTRACT

This paper was presented to the Sixth Workshop on Mathematical Foundations of Programming Semantics held at Queen's University, May 15-19, 1990.

The paper provides an introduction to a natural deduction based set theory, NaDSet, and illustrates its use in programming semantics. The need for such a set theory for the development of programming semantics is motivated by contrasting the presentation of recursive definitions within first order logic with their presentation within NaDSet. Within first order logic such definitions are always incomplete in a very simple sense: Induction axioms must be added to the given definitions and extended with every new recursive definition. Within a set theory such as NaDSet, recursive definitions of sets are represented as terms in the theory and are complete in the sense that all properties of the set can be derived from its definition. Such definitions not only have this advantage of completeness, but they also permit recursively defined sets to be members of the universe of discourse of the logic and thereby be shown to be members of other defined sets.

The resolution of the paradoxes provided by NaDSet is dependant upon replacing the naive comprehension axiom scheme of an inconsistent first order logic with natural deduction rules for the introduction of abstraction terms into arguments. The abstraction terms admitted are a generalization of the abstraction terms usually admitted into set theory. In order to avoid a confusion of use and mention, the nominalist interpretation of the atomic formulas of the logic forces NaDSet to be second order, although only a single kind of quantifier and variable is required.

The use of NaDSet for programming semantics is illustrated for a simple flow diagram language that has been used to illustrate the principles of denotational semantics. The presentation of the semantics within NaDSet is not only fully formal, in contrast to the simply mathematical presentation of denotational semantics, but because NaDSet is formalized as a natural deduction logic, its derivations can be simply checked by machine.

*Support of the Natural Science and Engineering Research Council of Canada is gratefully acknowledged.

1. INTRODUCTION

This paper provides an introduction to a natural deduction based set theory and logic NaDSet and illustrates its application to programming semantics. This introduction motivates the need for a new logic and gives a summary of NaDSet's distinguishing features. In section 2 the elementary syntax for the logic is described and illustrated with examples. In section 3 the logical syntax, or proof theory, for the logic is described. In section 4 one of the motivations for NaDSet, namely the "complete" nature of recursive definitions within it, is illustrated with some simple examples that also provide an introduction to the proof theory. In section 5, the semantics for a simple flow diagram language and a derivation of a simple theorem are provided within NaDSet. Some concluding remarks and future directions are given section 6.

1.1. Why a New Logic is Needed

Mathematics has traditionally used a process of abstraction to generalize and simplify structures: A property of objects is regarded as an object that may itself have properties. The traditional set theories are attempts to codify acceptable abstractions to ensure that undesirable conclusions are not drawn from sound premisses. But the concern of these set theories with what sets may correctly exist has given them an ad hoc character which may account for why "[they] have never been of particular interest to mathematicians. They now function mainly as a talismen to ward off evil "[Gray84]. This ad hoc character, as well as the complexity of their proof theory, make these set theories unsuitable for most uses within computer science.

The need for abstractions in computer science not available within traditional set theories has been argued many times. For example, [Scott70] describes the problems of self-application that can arise when interpreting programming languages and proposes a solution that has led to the development of denotational semantics. In Scott's foreword to [Stoy77], he concludes "For the future the problems of an adequqate proof theory and of explaining non-determinism loom very large." This quote can be interpreted as a call for a new formal logic within which the mathematical constructions of denotational semantics can be developed. But other needs for such a logic can be identified.

Horn clause programming, as introduced in Prolog, provides a computational model, but not a deductive model, for its programs. In NaDSet, the definition of a predicate by Horn clauses is an abstraction term that is complete in the sense that two predicates with different but equivalent definitions can be proved identical without additional axioms. In short, a proof theory for the semantics of Horn clause programming can be provided. For this reason, NaDSet may suggest

extensions to Prolog that incorporate second order concepts.

The increasing levels of abstraction required for the conceptual models used in enterprise modelling for database design, knowledge engineering and object-oriented systems, demands a logic within which such abstractions can be defined as objects and reasoned about. [Gilmore87a, 87b,88] describes applications of the earlier form of NaDSet to these problems.

Despite its widespread appeal, no suitable logic has been available within which category theory can be properly formalized [Feferman77,84]. As demonstrated in [Gilmore&Tsiknis90a,90b], category theory can be formalized within NaDSet, and a derivation provided for the theorem that the set of all categories is itself a category.

Finally, no suitable logic is available within which computational conjectures such as P≠NP may be explored. NaDSet may prove to be such a logic.

A logic that can satisfy the above demands must of necessity offer an elementary resolution of the paradoxes of set theory. In [Gilmore 71,80,86] an earlier version of the logic NaDSet was presented and a resolution of the paradoxes described. A consistency proof for the logic was provided. However, an extension to this logic is necessary for a proper formalization of category theory. Although NaDSet resolves the paradoxes in the same manner as the earlier logic, a consistency proof for NaDSet is not (yet) available.

1.2. Features of NaDSet

Classical first order logic provides a formalization of two of the three fundamental concepts of modern logic, namely truth functions and quantification. In classical set theories the third fundamental concept, namely abstraction, is formalized by adding axioms to first order logic. In NaDSet the three concepts are formalized in the same manner, namely through rules of deduction in a natural deduction presentation of the logic. This is the first of four distinguishing features of NaDSet which will be discussed.

1.2.1. Natural Deduction based Set Theory

Although the sequent calculus of [Gentzen34,35] is used for this paper, any natural deduction formalization of first order logic, such as those presented in [Beth55], [Prawitz65], or [Fitch 52] can be simply extended to be a formalization of NaDSet.

Natural deduction presentations of logic provide a transparent formalization of the traditional

reductionist semantics of [Tarski36], in which the truth value of a complex formula depends upon the truth values of simpler formulas, and eventually upon the truth values of atomic sentences. Formalizing abstractions in this way has the effect of replacing an unrestricted comprehension axiom scheme by a comprehension rule of deduction. This replacement is not novel to NaDSet; for example, several of the theories described in [Schütte60] or the set theory of Fitch described in [Prawitz65] or [Fitch 52] have this feature. This replacement is, however, not enough to ensure consistency; the theory described in [Gilmore68], for example, is inconsistent because of an improper definition of 'atomic formula'.

The interpretation of atomic formulas is critical for the reductionist semantics of Tarski. A second distinguishing feature of NaDSet is its interpretation of atomic formulas.

1.2.2. A Nominalist Interpretation of Atomic Formulas

In NaDSet, only names of sets, not sets may be members of sets. To emphasize that this interpretation is distinct from the interpretation of atomic formulas in classical set theory, ':' is used in place of 'ϵ' to denote the membership relationship. For example, the atomic formula

(i) {u | ~u:u }:C

is true in an interpretation if the term '{u | ~u:u }' is in the set assigned to 'C', and is false otherwise. Note that the term '{u | ~u:u }' is being mentioned in the formula while 'C' is being used.

To avoid confusions of use and mention warned against in [Tarski36] and [Church56], NaDSet must be in effect a second order logic. The first order domain for the logic is the set $\mathbb{D}$ of all closed terms in which no parameter ocurs, as defined in clause 4 of the definition of the elementary syntax in section 2. For example, the term '{u | ~u:u }' is a member of $\mathbb{D}$. The second order domain for the logic is the set of all subsets of $\mathbb{D}$. Thus if 'C' is a second order constant, then an interpretation will assign it a subset of $\mathbb{D}$, so that (i) will be true or false in the interpretation.

Although NaDSet is in effect a second order logic, the elementary syntax requires only one kind of quantifier used for quantification over both the first order and second order domains. This is the third distinguishing feature of NadSet

1.2.3. One Universal Quantifier Instead of Two

In classical logic, existential quantification can be defined in terms of universal quantification and negation for both first and second order quantifiers. This opportunity for simplification is exploited in NaDSet as well; but the elementary syntax requires only one universal quantifier, not

one for first order quantification and one for second order quantification. However, the second order nature of the logic is revealed in the two kinds of parameters that are required.

An occurrence of a parameter in a formula or term of NaDSet plays the role that it does in [Prawitz65], namely as the occurrence of a variable not bound by a quantifier or an abstraction term. The definition of a substitution operator for a term in which free variables occur is complicated by the possibility of a free occurrence of a variable in the term becoming bound after the substitution. Admitting parameters as free variables allows the simpler definition of a substitution operator restricted to terms without free occurrences of variables.

In an interpretation of NaDSet, first order parameters are assigned members of $\mathbb{D}$, while second order parameters are assigned subsets of $\mathbb{D}$.

1.2.4. A Generalized Abstraction

The term '{u | ~u:u }' introduced in 1.2.2 is a typical abstraction term for a set theory that admits such terms; they take the form {**v** | **F** }, where **v** is a variable, and **F** is a formula in which the variable may have a free occurrence. The term is understood to represent the set of **v** satisfying **F**. In NaDSet, however, **v** may be replaced by any term in which there is at least one free occurrence of a variable and there are no occurrences of parameters, as defined in clause 3 of the definition of elementary syntax in section 2. A term satisfying these conditions is the ordered pair term defined for variables **u** and **v** that are distinct from w as follows:

<u,v> for {w |(**u**:C ↓ **v**:C)}

Here '↓' is the single primitive logical connective of joint denial, in terms of which all other logical connectives are defined. That this simple term has the desired properties of the ordered pair is demonstrated in [Gilmore89]. The ordered pair term is used, for example, to define the Cartesian product of two sets **A** and **B**:

[AxB] for {<u,v> | (u:**A** ∧ v:**B**)}

The rules of deduction for the introduction of abstraction terms such as these are natural generalizations of the rules of deduction for abstraction terms of the form {**v** | **F**}. These abstraction rules determine what are appropriate uses of abstraction terms in mathematical arguments, rather than determine what sets may consistently coexist. For example, the arguments Russell used to show that the empty set is a member of the Russell set and that the universal set is not, are arguments that can be shown to be correct in NaDSet, while the arguments demonstrating that the Russell set is and is not a member of itself cannot be justified in NaDSet. Thus it can be said that NaDSet provides an answer to the question

What constitutes a sound argument?

rather than to the question

What sets exist?

which is a concern of the classical set theories. This is stressed in [Gilmore89] where it is demonstrated that the general diagonal argument of Cantor is not a sound argument, although the commonly used instances of it in computer science are sound.

2. ELEMENTARY SYNTAX OF NaDSet

The syntax requires distinguishing between five different kinds of sytactical objects, namely, variables, first and second order constants, and first and second order parameters. It is assumed that there are denumerably many objects of each kind.

Definition of Elementary Syntax

1.1. A variable is a term. The single occurrence of the variable in the term is a free occurrence in the term.

1.2. Any parameter or constant is a term. No variable has a free occurrence in the term.

2.1. If **ta** and **tb** are any terms, then **ta:tb** is a formula. A free occurrence of a variable in **ta** or in **tb**, is a free occurrence of the variable in the formula.

2.2. If **F** and **G** are formulas then (**F↓G**) is a formula. A free occurrence of a variable in **F** or in **G** is a free occurrence in (**F↓G**).

2.3. If **F** is a formula and **v** a variable, then ∀**vF** is a formula. A free occurrence of a variable other than **v** in **F**, is a free occurrence in ∀**vF**; no occurrence of **v** is free in ∀**vF**.

3. Let **ta** be any term in which there is at least one free occurrence of a variable and no occurrence of a parameter. Let **F** be any formula. Then {**ta**|**F**} is an abstraction term and a term. A free occurrence of a variable in **F** which does not also have a free occurrence in **ta**, is a free occurrence in {**ta**|**F**}. A variable with a free occurrence in **ta** has no free occurrence in {**ta**|**F**}.

4. A term is first order if no second order parameter occurs in it. A formula **t:T** is atomic if **t** is first order, and **T** is a second order parameter or constant. A term or formula in which no variable has a free occurrence is said to be closed.

It is important to understand what are free and bound occurrences of variables in a term {**ta**|**F**}.

Consider the formula:

$\langle u,v\rangle:\{\langle u,v\rangle \mid u:v \wedge \langle v,w\rangle:B\}$

The first occurrence of each of the variables 'u' and 'v' in this formula are free occurrences; all other occurrences of these variables are not free. The single occurrence of the variable 'w' is a free occurrence. Therefore in the formula

$[\forall u][\forall w](\langle u,v\rangle:\{\langle u,v\rangle \mid u:v \wedge \langle v,w\rangle:B\})$

only the first occurrence of 'v' is free.

Note that the first sentence of clause 4 applies to second order constants and to the abstraction terms defined in clause 3; for example, the second order constant 'B' is a first order term, as is also the abstraction term $\{\langle u,v\rangle \mid u:v \wedge \langle v,y\rangle:B\}$.

3. LOGICAL SYNTAX

NaDSet is presented as a Gentzen Sequent Calculus. Familiarity with the Gentzen sequent calculus as described in [Gentzen34,35], [Kleene52], or [Prawitz65] is presumed. As noted in the introduction, this natural deduction calculus is chosen for the formalization of NaDSet because it is one of the least complicated to describe and justify. However, any natural deduction formalization of first order logic, such as those presented in [Beth55], [Prawitz65], or [Fitch52], can be simply extended to be a formalization of NaDSet.

A sequent in NaDSet takes the form

$\Gamma \rightarrow \Theta,$

where Γ and Θ are finite, possibly empty, sequences of closed formulas. The formulas Γ form the **antecedent** of the sequent, and the formulas of Θ the **succedent**. A sequent can be interpreted as asserting that one of the formulas of its antecedent is false, or one of the formulas of its succedent is true.

3.1. Definition of Logical Syntax

Axioms

$\mathbf{G} \rightarrow \mathbf{G},$

where **G** is a closed atomic formula

Propositional Rules

$$\frac{\Gamma, G \to \Theta \qquad \Delta, H \to \Lambda}{\Gamma, \Delta \to (G{\downarrow}H), \Theta, \Lambda} \qquad \frac{\Gamma \to G, H, \Theta}{\Gamma, (G{\downarrow}H) \to \Theta}$$

Quantification Rules

$$\frac{\Gamma \to [p/u]F, \Theta}{\Gamma \to \forall uF, \Theta} \qquad \frac{\Gamma, [t/u]F \to \Theta}{\Gamma, \forall uF \to \Theta}$$

In the first rule, **p** is a parameter that does not occur in **F**, or in any formula of Γ or Θ.
In the second rule, **t** is any closed term.

Abstraction Rules

$$\frac{\Gamma \to [\underline{t}/\underline{u}]F, \Theta}{\Gamma \to [\underline{t}/\underline{u}]ta{:}\{ta|F\}, \Theta} \qquad \frac{\Gamma, [\underline{t}/\underline{u}]F \to \Theta}{\Gamma, [\underline{t}/\underline{u}]ta{:}\{ta|F\} \to \Theta}$$

$\underline{u}$ is a sequence of the distinct variables with free occurrences in the term **ta**.
F is a formula in which only the variables $\underline{u}$ have free occurrences.
$\underline{t}$ is a sequence of closed terms, one for each variable in $\underline{u}$.
$[\underline{t}/\underline{u}]$ is a substitution operator that replaces each occurrence of the variables $\underline{u}$, respectively, with the corresponding terms $\underline{t}$.

Structural Rules

The structural rules of [Gentzen34,35] as described in [Kleene52] consist of contraction rules, interchange rules and thinning rules. The contraction rules permit the removal of a duplicate formula in the antecedent or in the succedent of a sequent, the interchange rules permit changing the order of the formulas in the antecedent or succedent of a sequent, while the thinning rules permit the introduction of a new formula into the antedent or succedent of a sequent. The effect of the contraction and interchange rules is to treat the antecedent and succedent of a sequent as finite sets of formulas. For this reason these rules will be ignored in this paper.

Thinning rules

$$\frac{\Gamma \rightarrow \Theta}{\Gamma \rightarrow \mathbf{F}, \Theta} \qquad \frac{\Gamma \rightarrow \Theta}{\Gamma, \mathbf{F} \rightarrow \Theta}$$

where **F** is any closed formula.

Cut Rule

$$\frac{\Gamma \rightarrow \Theta, \mathbf{G} \qquad \mathbf{G}, \Delta \rightarrow \Lambda}{\Gamma, \Delta \rightarrow \Theta, \Lambda}$$

End of definition

Because the axioms are restricted to being sequents of closed formulas and the thinning rules may only introduce closed formulas, only sequents of closed formulas are derivable in NaDSet.

The propositional, quantification and abstraction rules will be denoted respectively by:

$\rightarrow\downarrow$, $\downarrow\rightarrow$, $\rightarrow\forall$, $\forall\rightarrow$, $\rightarrow\{\}$ and $\{\}\rightarrow$.

The thinning and cut rules will be referred to by name.

All the usual logical connectives $\sim$, $\wedge$, $\vee$, $\supset$ and $\equiv$ and the existential quantifier $\exists$ can be defined using $\downarrow$ and $\forall$. Corresponding rules of deduction can be derived and when necessary will be denoted respectively by:

$\sim\rightarrow$, $\rightarrow\sim$, $\rightarrow\wedge$, $\wedge\rightarrow$, $\rightarrow\vee$, $\vee\rightarrow$, $\rightarrow\supset$, $\supset\rightarrow$, $\rightarrow\equiv$, $\equiv\rightarrow$, $\rightarrow\exists$ and $\exists\rightarrow$.

Also rules of deduction can be derived for bounded quantifiers such as [$\forall$w:ExpCls].

Although not used in this paper, the abstraction terms of NaDSet permit another very useful bounded quantifier. The cartesian product of two sets **A** and **B** is defined:

(A x B) for $\{\langle u,v\rangle \mid (u:\mathbf{A} \wedge v:\mathbf{B}) \}$

Then a formula

$[\forall w:(\mathbf{A} \times \mathbf{B})]\mathbf{G}$

can be understood to be an abbreviation of a formula

$[\forall w1][\forall w2](\langle w1,w2\rangle:(\mathbf{A} \times \mathbf{B}) \supset [\langle w1,w2\rangle/w]\mathbf{G})$

where w1 and w2 are free to replace w in **F**. In general then a formula

$[\forall w:\{\mathbf{ta} \mid \mathbf{F}\}]\mathbf{G}$

is to be understood as an abbreviation for the formula:

$[\forall \underline{w}]([\underline{w}/\underline{u}]\mathbf{ta}:\{\mathbf{ta} \mid \mathbf{F}\} \supset [\,[\underline{w}/\underline{u}]\mathbf{ta}/w]\mathbf{G})$

Here $\underline{u}$ is a sequence of the distinct variables with free occurrences in the abstraction term **ta**, and $\underline{w}$ is a sequence of distinct variables of the same length free to replace the variables $\underline{u}$ in **G**.

The quantification rules require only one kind of universal quantifier, not the two of second order logic. The parameter appearing in the premiss of the $\rightarrow\forall$ rule, but not in its conclusion, will be either a first or a second order parameter, with the order of the parameter implicitly determining the order of the quantifier. There is not a similar restriction on the $\forall\rightarrow$ rule.

4. FORMALIZING RECURSIVE DEFINITIONS

Two contrasting approaches to formalizing recursive definitions correspond to two different views of mathematics. In the first, a derivative of Hilbert's "formalist" view of mathematics, a recursive definition is expressed by axioms added to first order logic. This is the method used in the programming language Prolog. In the second, a derivative of the Frege-Russell "logistic" view of mathematics [Wilder58], a recursive definition is provided by an abstraction term within a formalized set theory. For derivations of atomic sentences, that is for programming, both methods work equally well. But to prove results about programs, for example, to define a formal semantics for programs, only the second approach is satisfactory. This will be demonstrated in the next two subsections:

4.1. Axiomatic Method

Given 0 as a constant, and ' as a one place successor function, a one place natural number predicate N is defined by the axioms:

$N[0]$

$\forall u(\, N[u] \supset N[u'] \,)$

A related predicate NN can similarly be defined by the axioms

$NN[0']$

$\forall u(\, NN[u] \supset NN[u'] \,)$

However it is not possible to prove from these four axioms the theorem:

$\forall u(\, NN[u] \supset N[u] \,)$

A counter-example is provided by the interpretations

N is $\{\, 0, 0', 0'', \ldots \,\}$

NN is $\{c, c', c'', \ldots, 0', 0'', \ldots\}$

Under these interpretations, both N and NN satisfy their axioms, but

$(NN[c] \wedge \sim N[c])$

is true. To prove the theorem an additional axiom is needed for NN, namely

$(N[0'] \wedge \forall u(N[u] \supset N[u']) \supset \forall u(NN[u] \supset N[u]))$

which is an instance of an induction axiom for NN.

Because a recursive definition by first order axioms always requires the addition of new axioms with each new recursive definition, such definitions are said to be incomplete. Consider, for example, the definition of the plus predicate by the axioms:

$\forall u(N[u] \supset +[0,u,u])$

$\forall u \forall v \forall w(N[u] \wedge N[v] \wedge N[w] \wedge +[u,v,w] \supset +[u',v,w'])$

From the given axioms, the theorem

$\forall u(NN[u] \supset \exists v(NN[v] \wedge +[u,u,v])$

can only be proved from additional axioms for N.

The need to add additional axioms to recursive definitions is more than just an inconvenience: There is always the danger of adding inconsistent axioms. Further, some kinds of results for computer science, as for mathematics, require proving that a particular formula is not derivable; for such results it is necessary that all recursive definitions involved are complete.

4.2. Set Method

Consider now recusive definitions for N, NN and + in NaDSet. First intensional identity is defined:

$=$ **for** $\{<u,v> \mid \forall w (u:w \equiv v:w) \}$

The usual infix notation for identity is used in the following definitions:

0 **for** $\{u \mid \sim u = u \}$

$\{t\}$ **for** $\{v \mid v = t \}$

Cls **for** $\{z \mid \forall u(u:z \supset \{u\}:z) \}$

N **for** $\{x \mid [\forall z:Cls](0:z \supset x:z) \}$

NN **for** $\{x \mid [\forall z:Cls](\{0\}:z \supset x:z) \}$

The definition of Cls, prior to the definitions of N and NN, will be typical of the recursive definitions provided in this paper. Cls is the set of sets that are closed under successor. The bounded universal quantifier $[\forall z:Cls]$ in the definition of N, together with the scope of the quantifier, ensures that N is the least set, in which 0 is a member, closed under successor. The same quantifier in the definition of NN plays a similar role.

The sequent

$\rightarrow \forall u(\, u{:}NN \supset u{:}N\,)$

is derivable in NaDSet. In the following derivation from the sequents (a) and (b), 'p' and 'q' are first order parameters, and 'Q' is a second order parameter. Readers are advised to read the derivation as it has been developed, namely bottom-up. An attractive feature of natural deduction presentations is that the derivation of a sequent is <u>almost</u> determined by the sequent.

(a) $Q{:}Cls \rightarrow Q{:}Cls$

(b) $(\{0\}{:}Q \supset p{:}Q),\ Q{:}Cls \rightarrow (\, 0{:}Q \supset p{:}Q)$

$[\forall z{:}Cls](\, \{0\}{:}z \supset p{:}z),\ Q{:}Cls,\ Q{:}Cls \rightarrow (\, 0{:}Q \supset p{:}Q)$	$[\forall z{:}Cls]\rightarrow$
$p{:}NN,\ Q{:}Cls,\ Q{:}Cls \rightarrow (\, 0{:}Q \supset p{:}Q)$	$\{\}\rightarrow$
$p{:}NN,\ Q{:}Cls \rightarrow (\, 0{:}Q \supset p{:}Q)$	contraction $\rightarrow$
$p{:}NN \rightarrow [\forall z{:}Cls](\, 0{:}z \supset p{:}z)$	$\rightarrow[\forall z{:}Cls]$
$p{:}NN \rightarrow p{:}N$	$\rightarrow\{\}$
$\rightarrow \forall u(\, u{:}NN \supset u{:}N\,)$	$\rightarrow\forall$

A derivation of the sequent (a) follows:

$q{:}Q \rightarrow q{:}Q\ \ \{q\}{:}Q \rightarrow \{q\}{:}Q$	axioms

---

$q{:}Q,\ (q{:}Q \supset \{q\}{:}Q) \rightarrow \{q\}{:}Q$	$\supset\rightarrow$
$(q{:}Q \supset \{q\}{:}Q) \rightarrow (q{:}Q \supset \{q\}{:}Q)$	$\rightarrow\supset$
$[\forall u](u{:}Q \supset \{u\}{:}Q) \rightarrow (q{:}Q \supset \{q\}{:}Q)$	$\forall\rightarrow$
$[\forall u](u{:}Q \supset \{u\}{:}Q) \rightarrow [\forall u](u{:}Q \supset \{u\}{:}Q)$	$\rightarrow\forall$
$Q{:}Cls \rightarrow Q{:}Cls$	$\{\}\rightarrow, \rightarrow\{\}$

The need for a derivation of the sequent (a), which might be expected to be an axiom, is typical of NaDSet. In future, derivations of such sequents will be left as exercises for the reader.

A derivation of the sequent (b) follows:

$0{:}Q \rightarrow 0{:}Q\ \ p{:}Q \rightarrow p{:}Q\ \ \{0\}{:}Q \rightarrow \{0\}{:}Q$	axioms

---

$(\{0\}{:}Q \supset p{:}Q),\ (0{:}Q \supset \{0\}{:}Q),\ 0{:}Q \rightarrow p{:}Q$	$\supset\rightarrow, \supset\rightarrow$
$(\{0\}{:}Q \supset p{:}Q),\ (0{:}Q \supset \{0\}{:}Q) \rightarrow (\, 0{:}Q \supset p{:}Q)$	$\rightarrow\supset$
$(\{0\}{:}Q \supset p{:}Q),\ [\forall u](u{:}Q \supset \{u\}{:}Q) \rightarrow (\, 0{:}Q \supset p{:}Q)$	$\forall\rightarrow$
$(\{0\}{:}Q \supset p{:}Q),\ Q{:}Cls \rightarrow (\, 0{:}Q \supset p{:}Q)$	$\{\}\rightarrow$

Consider next +:

+Cls **for** {z | ∀u∀v∀w(<u,v,w>:z ⊃ <{u},v,{w}>:z) }

\+ **for** {<x,y,z> | [∀w:+Cls]([∀v] (v:N ⊃ <0,v,v>:+) ⊃ <x,y,z>:w) }

The following sequent can be derived in NaDSet:

→ [∀u:NN][∃v:NN](<u,u,v>:+)

As the reader can verify, a derivation can be provided directly from the given definitions, again emphasizing that logistic definitions are complete in themselves.

Recursive definitions in a set theory such as NaDSet have at least four important advantages over definitions by axioms. First, it is unnecessary to construct induction axioms, since they follow from the definitions. Although this is not difficult for the simple recursively defined sets described in [Manna&Waldinger84], complex recursive definitions, such as those provided in the next section for a simple flow diagram language, offer a challenge. Second, the definitions do not in any way modify the underlying logic, so that there need never be a concern that inconsistent axioms may be introduced. Third, the definitions are given as terms that can be shown to have properties and can be reasoned about, as demonstrated above for N and NN. But also these terms can be shown to be members of other sets; for example, each of N and NN can be shown to be a member of the universal set V1, that is defined to be {u | u=u }. Lastly, recursive definitions for disparate fields can be kept together without any concern that the axioms for one set of definitions will interact in unforseeable ways with those of another set of definitions. It is only necessary to maintain a discipline which ensures that the abbreviating name for a term uniquely identifies the term, at least in the contexts in which the term will be used.

These advantages suggest that NaDSet may provide extensions to Horn clause programming languages such as Prolog.

5. EXAMPLE OF PROGRAMMING SEMANTICS

In this section, semantics will be defined within NaDSet for the simple language of flow diagrams used in [Stoy77] as an example of the application of denotational semantics. Since they add nothing to the exposition here, primitive commands and primitive predicates will not be considered.

5.1. Syntax

The elementary syntax for the language is described in a variant of Backus-Naur form. First the set of expressions is defined:

Exp ::= true | false | < Exp_0, Exp_1, Exp_2, CndExp >

Here true, false and CndExp are given constant strings, with the latter abbreviating 'conditional expression'. This Backus-Naur form is, of course, a recursive definition of a set Exp. In NaDSet it would be defined:

Exp **for** {e | [∀z](true:z ∧ false:z ∧ [∀u,v,w:z]<u,v,w,CndExp>:z ⊃ e:z)}

The Backus-Naur form may be regarded as a 'sugaring' of this definition.

The set of commands is similarly defined:

Cmd ::= dummy | < Exp, Cmd_1, Cmd_2, CndCmd > | < Cmd_1, Cmd_2, SeqCmd > |

< Exp, Cmd, WCmd > | < Cmd, Exp, RWCmd >

The five nondummy commands are respectively the conditional command, the sequence command, the whiledo command and the repeatwhile command. As with the set Exp, the set Cmd could be defined in NaDSet.

The strings false, true, CndExp, dummy, CndCmd, SeqCmd, WCmd, and RWCmd are the primitive strings of the language. They are not only assumed to be distinct, but must be assumed to be provably distinct. By this is meant that for each distinct pair **St1** and **St2** of strings from this list, the following sequent is derivable in NaDSet:

St1 = **St2** →

One of the simplest ways of assuring this is to take the strings in order as abbreviations of the integers 0, 1, 2, 3, 4, 5, 6, 7, since all integers can be proved to be distinct using only the definition N of the integers.

In addition, the basic properties of ordered pairs and tuples, shown to be derivable in [Gilmore89], must be assumed. In particular, therefore, the following sequent is derivable:

<t1,t2,SeqCmd> = <t3,t4,WCmd> →

for any first order terms t1, t2, t3, and t4.

5.2. Expression Semantics

It is assumed that a finite set S of states has been defined, and that the set B has been defined

B **for** {v | v = 1 ∨ v = 0}

to represent the set of boolean values.

A set ExpSem is to be defined so that "<e,s,v>:ExpSem" means "expression e in state s has value v". But first a definition of ExpCls, expression closed, is needed:

ExpCls **for** {w | [∀e_0, e_1,e_2:Exp][∀r:S][∀v:B](

$(\langle e_0,r,1\rangle{:}w \wedge \langle e_1,r,v\rangle{:}w \supset \langle\langle e_0,e_1,e_2,\underline{CndExp}\rangle,r,v\rangle{:}w)$

$\wedge\ (\langle e_0,r,0\rangle{:}w \wedge \langle e_2,r,v\rangle{:}w \supset \langle \langle e_0,e_1,e_2,\underline{CndExp}\rangle,r,v\rangle{:}w)\ )\ \}$

ExpSem **for** $\{\langle e,r,v\rangle \mid [\forall w{:}ExpCls]([\forall r{:}S]\langle \underline{true}, r,1\rangle{:}w \wedge [\forall r{:}S]\langle \underline{false}, r,0\rangle{:}w \supset \langle e,r,v\rangle{:}w)\}$

5.3. Command Semantics

Now a set CmdSem is to be defined so that "<c,r,s>:CmdSem" means "command c in state r moves system to state s". Command closed is defined first:

CmdCls **for** $\{w \mid$

$[\forall c_1,c_2{:}Cmd][\forall r,s,t{:}S](\langle c_1,r,s\rangle{:}w \wedge \langle c_2,s,t\rangle{:}w \supset \langle\langle c_1,c_2,\underline{SeqCmd}\rangle,r,t\rangle{:}w)$

$\wedge\ [\forall e{:}Exp][\forall c_1,c_2{:}Cmd][\forall r,s{:}S](\langle e,r,1\rangle{:}ExpSem \wedge \langle c_1,r,s\rangle{:}w \supset \langle\langle e,c_1,c_2,\underline{CndCmd}\rangle,r,s\rangle{:}w\ )$

$\wedge\ [\forall e{:}Exp][\forall c_1,c_2{:}Cmd][\forall r,s{:}S](\langle e,r,0\rangle{:}ExpSem \wedge \langle c_2,r,s\rangle{:}w \supset \langle\langle e,c_1,c_2,\underline{CndCmd}\rangle,r,s\rangle{:}w\ )$

$\wedge\ [\forall e{:}Exp][\forall c{:}Cmd][\forall r{:}S](\langle e,r,0\rangle{:}ExpSem \supset \langle\langle e,c,\underline{WCmd}\rangle,r,r\rangle{:}w)$

$\wedge\ [\forall e{:}Exp][\forall c{:}Cmd][\forall r,s,t{:}S](\langle e,r,1\rangle{:}ExpSem \wedge \langle c,r,s\rangle{:}w \supset (\langle\langle e,c,\underline{WCmd}\rangle,s,t\rangle{:}w \supset \langle\langle e,c,\underline{WCmd}\rangle,r,t\rangle{:}w\ ))$

$\wedge\ [\forall e{:}Exp][\forall c{:}Cmd][\forall r,s{:}S](\langle e,s,0\rangle{:}ExpSem \wedge \langle c,r,s\rangle{:}w \supset \langle\langle c,e,\underline{RWCmd}\rangle,r,s\rangle{:}w\ )$

$\wedge\ [\forall e{:}Exp][\forall c{:}Cmd][\forall r,s,t{:}S](\langle e,s,1\rangle{:}ExpSem \wedge \langle c,r,s\rangle{:}w \supset (\langle\langle c,e,\underline{RWCmd}\rangle,s,t\rangle{:}w \supset \langle\langle c,e,\underline{RWCmd}\rangle,r,t\rangle{:}w\ ))\}$

CmdSem **for**

$\{\langle c,r,s\rangle \mid [\forall w{:}CmdCls]([\forall t{:}S]\langle \underline{dummy},t,t\rangle{:}w \supset \langle c,r,s\rangle{:}w\ )\}$

5.4. Example Theorem

To illustrate the point that all the desired properties of the set CmdSem can be derived from its definition and the definitions of the other sets upon which it is dependant, a sketch of a derivation of the following sequent will be provided:

(th) $\rightarrow [\forall e{:}Exp][\forall c{:}Cmd][\forall r,t{:}S]\ ($

$\langle\langle c,e,\underline{RWCmd}\rangle,r,t\rangle{:}CmdSem \equiv \langle\langle c,\langle e,c,\underline{WCmd}\rangle,\underline{SeqCmd}\rangle,r,t\rangle{:}CmdSem\ )$

This sequent expresses that the effect of the command <c,e,RWCmd> is the same as the effect of

the sequence of commands c and <e,c,WCmd> in terms of state transformations. It is theorem 9.22 of [Stoy77].

It is interesting to contrast the definition of CmdSem with the denotational semantics for the commands provided in [Stoy77]. Apart from the obvious absence of the top and bottom in the definition of CmdSem, the most striking difference is that the semantics for all commands of the language is provided in the definitions of CmdCls and CmdSem, while the semantics for the commands are provided separately by Stoy. However, the properties of the separate commands can be recovered using the following parameterized definitions:

$$\mathbf{WCls[e,c]}\ \ \mathbf{for}\ \ \{w \mid [\forall r{:}S](\ \langle e,r,0\rangle{:}\mathrm{ExpSem} \supset \langle r,r\rangle{:}w\ ) \wedge [\forall r,s,t{:}S]\ (\langle e,r,1\rangle{:}\mathrm{ExpSem} \wedge \langle c,r,s\rangle{:}\mathrm{CmdSem} \wedge \langle s,t\rangle{:}w \supset \langle r,t\rangle{:}w\ )\}$$

$$\mathbf{RWCls[c,e]}\ \ \mathbf{for}\ \ \{w \mid [\forall r,t{:}S](\ \langle e,t,0\rangle{:}\mathrm{ExpSem} \wedge \langle c,r,t\rangle{:}\mathrm{CmdSem} \supset \langle r,t\rangle{:}w\ ) \wedge [\forall r,s,t{:}S]\ (\langle e,s,1\rangle{:}\mathrm{ExpSem} \wedge \langle c,r,s\rangle{:}\mathrm{CmdSem} \wedge \langle s,t\rangle{:}w \supset \langle r,t\rangle{:}w\ )\}$$

The properties of the separate commands are summarized in the following lemma:

5.4.1. Lemma.

The following sequents are derivable.

(1) $\rightarrow [\forall t{:}S]\ \langle \underline{\mathrm{dummy}},t,t\rangle{:}\mathrm{CmdSem}$

(2) $\rightarrow [\forall c_1,c_2{:}\mathrm{Cmd}][\forall r,t{:}S](\ \langle\langle c_1,c_2,\underline{\mathrm{SeqCmd}}\rangle,r,t\rangle{:}\mathrm{CmdSem} \equiv [\exists s{:}S](\ \langle c_1,r,s\rangle{:}\mathrm{CmdSem} \wedge \langle c_2,s,t\rangle{:}\mathrm{CmdSem}\ ))$

(3) $\rightarrow [\forall e{:}\mathrm{Exp}][\forall c_1,c_2{:}\mathrm{Cmd}][\forall r,s{:}S](\ \langle\langle e,c_1,c_2,\underline{\mathrm{CndCmd}}\rangle,r,s\rangle{:}\mathrm{CmdSem} \equiv (\langle e,r,1\rangle{:}\mathrm{ExpSem} \wedge \langle c_1,r,s\rangle{:}\mathrm{CmdSem}) \vee (\ \langle e,r,0\rangle{:}\mathrm{ExpSem} \wedge \langle c_2,r,s\rangle{:}\mathrm{CmdSem}\ ))$

(4) $\rightarrow [\forall e{:}\mathrm{Exp}][\forall c{:}\mathrm{Cmd}][\forall r,t{:}S](\ \langle\langle e,c,\underline{\mathrm{WCmd}}\rangle,r,t\rangle{:}\mathrm{CmdSem} \equiv [\forall z{:}\mathrm{WCls}[e,c]]\ \langle r,t\rangle{:}z\ )$

(5) $\rightarrow [\forall e{:}\mathrm{Exp}][\forall c{:}\mathrm{Cmd}][\forall r,t{:}S](\ \langle\langle c,e,\underline{\mathrm{RWCmd}}\rangle,r,t\rangle{:}\mathrm{CmdSem} \equiv [\forall z{:}\mathrm{RWCls}[c,e]]\ \langle r,t\rangle{:}z\ )$

The proof of this lemma has been placed in the appendix so as not to interupt the derivation of the sequent (th).

5.4.2. Derivation of Sequent (th)

Using the variables in the formula of the sequent as first order parameters, the sequent can be

simply derived from the following two sequents:

(i) e:Exp, c:Cmd, r:S, t:S, <<c,e,RWCmd>,r,t>:CmdSem
→ <<c,<e,c,WCmd>,SeqCmd>,r,t>:CmdSem

(ii) e:Exp, c:Cmd, r:S, t:S, <<c,<e,c,WCmd>,SeqCmd>,r,t>:CmdSem
→ <<c,e,RWCmd>,r,t>:CmdSem

Sketches of a derivation of the first sequent will be provided in the next subsections. The derivation of the second sequent is similar and therefore is omitted. As before the reader is advised to read the derivations as they have been developed, namely bottom-up. But note that the derivations are not complete; several applications of rules of deduction may be presented as one step in the derivation. To assist the reader in following the derivation, an asterisk * prefixes the formula that is the principle formula of the last rule applied.

5.4.2.1. Derivation of (i)

The following abbreviation will be used in this subsection only:

T **for** {<r,t> | [∃s1:S](<c,r,s1>:CmdSem ∧ <<c,<e,c,WCmd>,s1,t>:CmdSem)}

There follows a sketch of a derivation of (i) from two sequents (a) and (b) in which this abbreviation has been used. In the derivation, and in all subsequent derivations, a long bar will be used to indicate the application of one or more rules with two premisses. A short bar is used to indicate the beginning of the list of premisses for the next long bar application. If no short bar appears between two long bars, then all sequents between the two long bars are premisses for the next long bar application. Unless a premiss is numbered, only the first premiss in a list of premisses for a long bar application is provided with a derivation. The derivation appears immediately above the first premiss. Derivations of premisses not provided are left as exercises for the reader; for example, a derivation for a premiss of the form Γ, **F** → **F** is left as an exercise. Rules that are applied in the derivation are not identified; instead the principal formula of the conclusion is prefixed with *.

(a) e:Exp, c:Cmd → T:RWCls[c,e]

(b) e:Exp, c:Cmd, r:S, t:S, <r,t>:T
→ [∃s1:S](<c,r,s1>:CmdSem ∧ < <c,<e,c,WCmd>,s1,t>:CmdSem)

---

e:Exp, c:Cmd, r:S, t:S, *(T:RWCls[c,e] ⊃ <r,t>:T)
→ [∃s1:S](<c,r,s1>:CmdSem ∧ < <c,<e,c,WCmd>,s1,t>:CmdSem)

e:Exp, c:Cmd, r:S, t:S, *[∀z:RWCls[c,e]] <r,t>:z
→ [∃s1:S](<c,r,s1>:CmdSem ∧ < <c,<e,c,WCmd>,s1,t>:CmdSem)

[Using sequents (2) and (5) of lemma 5.4.1 and cut rule]

e:Exp, c:Cmd, r:S, t:S, *<<c,e,RWCmd>,r,t>:CmdSem
→ *<<c,<e,c,WCmd>,SeqCmd>,r,t>:CmdSem

A derivation of sequent (b) will be left as an exercise for the reader. A derivation of sequent (a) follows.

e:Exp, r:S, s:S, <e,s,1>:ExpSem → <e,s,1>:ExpSem
c:Cmd, s:S, s1:S, <c,s,s1>:CmdSem → <c,s,s1>:CmdSem
t:S, s1:S, <s1,t>:W → <s1,t>:W

---

e:Exp, c:Cmd, r:S, s:S, t:S, s1:S, <e,s,1>:ExpSem, <c,s,s1>:CmdSem, <s1,t>:W,
→ *(<e,s,1>:ExpSem ∧ <c,s,s1>:CmdSem ∧ <s1,t>:W)
<s,t>:W → <s,t>:W

---

e:Exp, c:Cmd, r:S, s:S, t:S, s1:S, <e,s,1>:ExpSem, <c,s,s1>:CmdSem, <s1,t>:W,
*(<e,s,1>:ExpSem ∧ <c,s,s1>:CmdSem ∧ <s1,t>:W ⊃ <s,t>:W)
→ <s,t>:W
s:S → s:S
s1:S → s1:S
t:S → t:S

---

e:Exp, c:Cmd, r:S, s:S, t:S, s1:S, <e,s,1>:ExpSem, <c,s,s1>:CmdSem, <s1,t>:W,
*[∀r,s,t:S] (<e,r,1>:ExpSem ∧ <c,r,s>:CmdSem ∧ <s,t>:W ⊃ <r,t>:W)
→ <s,t>:W

---

e:Exp, c:Cmd, r:S, s:S, t:S, s1:S, <e,s,1>:ExpSem, <c,s,s1>:CmdSem, <s1,t>:W,
*W:WCls[e,c]
→ <s,t>:W (thinning, ∧→, {}→)
W:WCls[e,c] → W:WCls[e,c]

---

e:Exp, c:Cmd, r:S, s:S, t:S, s1:S, <e,s,1>:ExpSem, <c,s,s1>:CmdSem,
*[∀z:WCls[e,c]] <s1,t>:z , W:WCls[e,c]
→ <s,t>:W
e:Exp, c:Cmd, r:S, s:S, t:S, s1:S, <e,s,1>:ExpSem, <c,s,s1>:CmdSem,
[∀z:WCls[e,c]] <s1,t>:z
→ *[∀z:WCls[e,c]] <s,t>:z

---

e:Exp, c:Cmd, r:S, s:S, t:S, s1:S, <e,s,1>:ExpSem, <c,s,s1>:CmdSem,
*<<c,<e,c,WCmd>,s1,t>:CmdSem
→ *<<c,<e,c,WCmd>,s,t>:CmdSem
[Using sequent (4) in lemma 5.4.1 and cut]
c:Cmd, r:S, s:S, <c,r,s>:CmdSem → <c,r,s>:CmdSem
s:S → s:S

---

e:Exp, c:Cmd, r:S, s:S, t:S, s1:S, <e,s,1>:ExpSem, <c,r,s>:CmdSem,
<c,s,s1>:CmdSem, <<c,<e,c,WCmd>,s1,t>:CmdSem
→ *(s:S ∧ <c,r,s>:CmdSem ∧ <<c,<e,c,WCmd>,s,t>:CmdSem)
e:Exp, c:Cmd, r:S, s:S, t:S, s1:S, <e,s,1>:ExpSem, <c,r,s>:CmdSem,
<c,s,s1>:CmdSem, <<c,<e,c,WCmd>,s1,t>:CmdSem
→ *[∃s1:S](<c,r,s1>:CmdSem ∧ <<c,<e,c,WCmd>,s1,t>:CmdSem)
e:Exp, c:Cmd, r:S, s:S, t:S, <e,s,1>:ExpSem, <c,r,s>:CmdSem,
[∃s1:S](<c,s,s1>:CmdSem ∧ <<c,<e,c,WCmd>,s1,t>:CmdSem)
→ [∃s1:S](<c,r,s1>:CmdSem ∧ <<c,<e,c,WCmd>,s1,t>:CmdSem)
e:Exp, c:Cmd, r:S, s:S, t:S, <e,s,1>:ExpSem, <c,r,s>:CmdSem,
*<s,t>:T → *<r,t>:T

---

e:Exp, c:Cmd →
*[∀r,s,t:S] *(<e,s,1>:ExpSem ∧ <c,r,s>:CmdSem ∧ <s,t>:T ⊃ <r,t>:T)
Similarly
e:Exp, c:Cmd →
[∀r,t:S](<e,t,0>:ExpSem ∧ <c,r,t>:CmdSem ⊃ <r,t>:T)

---

e:Exp, c:Cmd →
*([∀r,t:S](<e,t,0>:ExpSem ∧ <c,r,t>:CmdSem ⊃ <r,t>:T)
∧ [∀r,s,t:S] (<e,s,1>:ExpSem ∧ <c,r,s>:CmdSem ∧ <s,t>:T ⊃ <r,t>:T))
e:Exp, c:Cmd → *T:RWCls[c,e]
End of derivation of sequent (i)
End of derivation of sequent (th)

6. FUTURE WORK

Recursive definitions have been used to define a formal semantics of a simple programming language within NaDSet. The set of expressions, commands and their semantics are given by NaDSet terms that formalize specific recursive definitions. A theorem that expresses the equivalence between two type of commands has been derived from the definitions alone without a need for any type of fixed point induction and admissibility tests [Stoy 77]. The domains needed for the semantics are implicitly defined. But the first goal set for denotational semantics by Scott was the development of a semantics for the lambda calculus [Church41]. Because of the nominalist interpretation of atomic formulas in NaDSet, it is not difficult to give definitions for lambda abstraction and application within NaDSet [Gilmore&Tsiknis90c]. The question as to whether these definitions lead to an interpretation of the lambda calculus remains open, as does also the related question as to what function spaces can be constructed within NaDSet.

BIBLIOGRAPHY

The numbers in parentheses refer to the date of publication. (xx) is the year 19xx.

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.

Church, Alonzo

(41) The Calculi of Lambda Conversion, Princeton University Press.

(56) Introduction to Mathematical Logic, Vol I, Princeton University Press.

Feferman, Solomon

(77) Categorical Foundations and Foundations of Category Theory, Logic, Foundations of Mathematics and Computability Theory, Editors Butts and Hintikka, D. Reidel, 149-169.

(84) Towards Useful Type-Free Theories, I, *Journal of Symbolic Logic*, March, 75-111.

Fitch, Frederick B.

(52) Symbolic Logic: An Introduction, Ronald Press, New York.

Gentzen, Gerhard

(34,35) Untersuchungen über das logische Schliessen, *Mathematische Zeitschrift*, 39, 176-210, 405-431.

Gilmore, P.C. (Paul C.),

(68) A Formalized Naive Set Theory, a paper presented at the Summer Conference on Intuitionism and Proof Theory, Buffalo, New York.

(71) A Consistent Naive Set Theory: Foundations for a Formal Theory of Computation, IBM Research Report RC 3413, June 22.

(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. This is a revised version of [Gilmore71].

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

(87a) The SET Conceptual Model and the Domain Graph Method of Table Design, UBC Computer Science Department Technical Report 87-7.

(87b) Justifications and Applications of the SET Conceptual Model, UBC Computer Science Department Technical Report 87-9.

(88) A Foundation for the Entity Relationship Approach: How and Why, Proceedings of the 6th Entity Relationship Conference, editor S.T. March, North-Holland 95-113.

(89) How Many Real Numbers are There?, UBC Technical Report TR 89-7, revised July 1990.

Gilmore, Paul C. & Tsiknis, George K.

(90a) A Logic for Category Theory, UBC Computer Science Department Technical Report 90-2, May, 1990.

(90b) 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. UBC Computer Science Department Technical Report 90-23.

(90c) Function Spaces in NaDSet, in preparation.

Gray, J.W.

(84) Editor, Mathematical Applications of Category Theory, *Contemporary Mathematics*, 30, American Mathematical Society.

Manna, Zohar & Waldinger, Richard

(84) The Logical Basis for Computer Programming, Volume 1: Deductive Reasoning, Addison-Wesley

Prawitz, Dag

(65) Natural Deduction, A Proof-Theoretical Study, Stockholm Studies in Philosphy 3, Almquist & Wiksell, Stockholm.

Schütte, K

(60) Beweisetheorie, Springer.

Stoy, Joseph E.

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

Tarski, Alfred

(36) Der Wahrheitsbegriff in den formalisierten Sprachen, *Studia Philosophica*, vol. 1. pp261-405. English translation appears in Logic, Semantics, Metamathematics, Papers from 1923 to 1938, Oxford University Press, 152-278, 1956.

Tsiknis, George K.

(90) Applications of a Natural Deduction Based Set Theory, UBC Department of Computer Science PhD thesis. In preparation.

Wilder, Raymond L.

(58) Introduction to the Foundations of Mathematics, Wiley

APPENDIX

Recall the definitions:

WCls[e,c] **for** {w | [∀r:S](<e,r,0>:ExpSem ⊃ <r,r>:w)
∧ [∀r,s,t:S] (<e,r,1>:ExpSem ∧ <c,r,s>:CmdSem ∧ <s,t>:w ⊃ <r,t>:w)}

RWCls[c,e] **for** {w | [∀r,t:S](<e,t,0>:ExpSem ∧ <c,r,t>:CmdSem ⊃ <r,t>:w)
∧ [∀r,s,t:S] (<e,s,1>:ExpSem ∧ <c,r,s>:CmdSem ∧ <s,t>:w ⊃ <r,t>:w)}

and the statement of the lemma:

Lemma 5.4.1.

The following sequents are derivable.

(1) → [∀t:S] <dummy,t,t>:CmdSem

(2) → [∀c_1,c_2:Cmd][∀r,t:S](<< c_1,c_2,SeqCmd>,r,t>:CmdSem
≡[∃s:S](<c_1,r,s>:CmdSem ∧ <c_2,s,t>:CmdSem))

(3) → [∀e:Exp][∀c_1,c_2:Cmd][∀r,s:S](<<e,c_1,c_2,CndCmd>,r,s>:CmdSem
≡ (<e,r,1>:ExpSem ∧ <c_1,r,s>:CmdSem) ∨ (<e,r,0>:ExpSem ∧ <c_2,r,s>:CmdSem))

(4) → [∀e:Exp][∀c:Cmd][∀r,t:S](<<e,c,WCmd>,r,t>:CmdSem
≡ [∀z:WCls[e,c]] <r,t>:z)

(5) → [∀e:Exp][∀c:Cmd][∀r,t:S](<<c,e,RWCmd>,r,t>:CmdSem
≡ [∀z:RWCls[c,e]] <r,t>:z)

Proof of lemma 5.4.1

Only the proof for case (4) is given. The other cases are similar. The proof consists of two parts, each deriving one of the following sequents:

(a) → [∀e:Exp][∀c:Cmd][∀r,t:S]([∀z:WCls[e,c]] <r,t>:z ⊃ <<e,c,WCmd>,r,t>:CmdSem)

(b) → [∀e:Exp][∀c:Cmd][∀r,t:S](<<e,c,WCmd>,r,t>:CmdSem ⊃ [∀z:WCls[e,c]] <r,t>:z)

1. A derivation of sequent (a):

<<e,c,WCmd>,s,t>:W → <<e,c,WCmd>,s,t>:W
<<e,c,WCmd>,r,t>:W → <<e,c,WCmd>,r,t>:W

---

*(<<e,c,WCmd>,s,t>:W ⊃ <<e,c,WCmd>,r,t>:W), <<e,c,WCmd>,s,t>:W
→ <<e,c,WCmd>,r,t>:W
<e,r,1>:ExpSem → <e,r,1>:ExpSem

W:CmdCls, <c,r,s>:CmdSem → <c,r,s>:W

W:CmdCls,
*(<e,r,1>:ExpSem ∧ <c,r,s>:W ⊃ (<<e,c,WCmd>,s,t>:W ⊃ <<e,c,WCmd>,r,t>:W)),
<e,r,1>:ExpSem, <c,r,s>:CmdSem, <<e,c,WCmd>,s,t>:W
→ <<e,c,WCmd>,r,t>:W

r:S → r:S
s:S → s:S
t:S → t:S

---

r:S, s:S, t:S, W:CmdCls,
*[∀r,s,t:S](<e,r,1>:ExpSem ∧ <c,r,s>:W ⊃ (<<e,c,WCmd>,s,t>:W
⊃ <<e,c,WCmd>,r,t>:W)),
<e,r,1>:ExpSem, <c,r,s>:CmdSem, <<e,c,WCmd>,s,t>:W
→ <<e,c,WCmd>,r,t>:W

W:CmdCls,
[∀r,s,t:S](<e,r,1>:ExpSem ∧ <c,r,s>:W ⊃ (<<e,c,WCmd>,s,t>:W
⊃ <<e,c,WCmd>,r,t>:W))
→ *[∀r,s,t:S]*(<e,r,1>:ExpSem ∧ <c,r,s>:CmdSem ∧ <<e,c,WCmd>,s,t>:W
⊃ <<e,c,WCmd>,r,t>:W)
[∀r:S](<e,r,0>:ExpSem ⊃ <<e,c,WCmd>,r,r>:W)
→ [∀r:S](<e,r,0>:ExpSem ⊃ <<e,c,WCmd>,r,r>:W)

W:CmdCls,
[∀r:S](<e,r,0>:ExpSem ⊃ <<e,c,WCmd>,r,r>:W) ,
[∀r,s,t:S](<e,r,1>:ExpSem ∧ <c,r,s>:W
⊃ (<<e,c,WCmd>,s,t>:W ⊃ <<e,c,WCmd>,r,t>:W))
→ *([∀r:S](<e,r,0>:ExpSem ⊃ <<e,c,WCmd>,r,r>:W)
∧ [∀r,s,t:S] (<e,r,1>:ExpSem ∧ <c,r,s>:CmdSem
∧ <<e,c,WCmd>,s,t>:W ⊃ <<e,c,WCmd>,r,t>:W))
e:Exp → e:Exp
c:Cmd → c:Cmd
e:Exp → e:Exp
c:Cmd → c:Cmd

---

e:Exp, c:Cmd, W:CmdCls,
[∀e:Exp][∀c:Cmd][∀r:S](<e,r,0>:ExpSem ⊃ <<e,c,WCmd>,r,r>:W) ,
[∀e:Exp][∀c:Cmd][∀r,s,t:S](<e,r,1>:ExpSem ∧ <c,r,s>:W
⊃ (<<e,c,WCmd>,s,t>:W ⊃ <<e,c,WCmd>,r,t>:W))
→ *([∀r:S](<e,r,0>:ExpSem ⊃ <<e,c,WCmd>,r,r>:W)
∧ [∀r,s,t:S] (<e,r,1>:ExpSem ∧ <c,r,s>:CmdSem
∧ <<e,c,WCmd>,s,t>:W ⊃ <<e,c,WCmd>,r,t>:W))

e:Exp, c:Cmd, W:CmdCls,
[∀e:Exp][∀c:Cmd][∀r:S](<e,r,0>:ExpSem ⊃ <<e,c,WCmd>,r,r>:W),
[∀e:Exp][∀c:Cmd][∀r,s,t:S](<e,r,1>:ExpSem ∧ <c,r,s>:W
⊃ (<<e,c,WCmd>,s,t>:W ⊃ <<e,c,WCmd>,r,t>:W))
→ ([∀r:S](<e,r,0>:ExpSem ⊃ *<r,r>:{<x,y> |
<<e,c,WCmd>,x,y>:W})
∧ [∀r,s,t:S] (<e,r,1>:ExpSem ∧ <c,r,s>:CmdSem
∧ *<s,t>:{<x,y>| <<e,c,WCmd>,x,y>:W}
⊃ *<r,t>:{<x,y>| <<e,c,WCmd>,x,y>:W}))

---

e:Exp, c:Cmd, *W:CmdCls
→ *{<x,y>| <<e,c,WCmd>,x,y>:W}:WCls[e,c]
e:Exp, c:Cmd, r:S, t:S, <r,t>:{<x,y>| <<e,c,WCmd>,x,y>:W}
→ <<e,c,WCmd>,r,t>:W

---

e:Exp, c:Cmd, r:S, t:S, W:CmdCls,
*({<x,y>| <<e,c,WCmd>,x,y>:W}:WCls[e,c]
⊃ <r,t>:{<x,y>| <<e,c,WCmd>,x,y>:W})
→ <<e,c,WCmd>,r,t>:W
e:Exp, c:Cmd, r:S, t:S, *[∀z:WCls[e,c]] <r,t>:z, W:CmdCls
→ <<e,c,WCmd>,r,t>:W [thinning]
e:Exp, c:Cmd, r:S, t:S, [∀z:WCls[e,c]] <r,t>:z,
W:CmdCls, [∀s1:S]<dummy,s1,s1>:W
→ <<e,c,WCmd>,r,t>:W
e:Exp, c:Cmd, r:S, t:S, [∀z:WCls[e,c]] <r,t>:z
→ *[∀w:CmdCls] *([∀s1:S]<dummy,s1,s1>:w ⊃ <<e,c,WCmd>,r,t>:w)
e:Exp, c:Cmd, r:S, t:S, [∀z:WCls[e,c]] <r,t>:z
→ *<<e,c,WCmd>,r,t>:CmdSem
→ *[∀e:Exp]*[∀c:Cmd]* [∀r,t:S]*([∀z:WCls[e,c]] <r,t>:z
⊃ <<e,c,WCmd>,r,t>:CmdSem)

2. A derivation of sequent (b)

There follows a derivation of (b) from the sequents (c) and (d). In this derivation, the following abbreviation is used:

T **for** {<x,y,z>| <x,y,z>:CmdSem ∧ (x=<e,c,WCmd> ⊃ <y,z>:W)}

(c) e:Exp, c:Cmd, W:WCls[e,c] → T:CmdCls

(d) e:Exp, c:Cmd, r:S, t:S,
([∀s1:S]<dummy,s1,s1>:T ⊃ <<e,c,WCmd>,r,t>:T)
→ <r,t>:W

---

e:Exp, c:Cmd, r:S, t:S, W:WCls[e,c],
*[∀w:CmdCls] ([∀s1:S]<dummy,s1,s1>:w ⊃ <<e,c,WCmd>,r,t>:w)
→ <r,t>:W
e:Exp, c:Cmd, r:S, t:S, *<<e,c,WCmd>,r,t>:CmdSem, W:WCls[e,c]
→ <r,t>:W
e:Exp, c:Cmd, r:S, t:S, <<e,c,WCmd>,r,t>:CmdSem
→ *[∀z:WCls[e,c]] <r,t>:z
→ *[∀e:Exp]*[∀c:Cmd] *[∀r,t:S]*(<<e,c,WCmd>,r,t>:CmdSem
⊃ [∀z:WCls[e,c]] <r,t>:z)

A derivation of sequent (d) will be left as an exercize for the reader.

2.1. A derivation of sequent (c)

Let [T/w]Si, $1 \le i \le 7$, be the formula obtained by replacing w by T in the i-th conjunct in the definition of CmdSem. Then, (c) is obtained from the derivations of

(ci) e:Exp, c:Cmd, W:WCls[e,c] → [T/w]Si

for $1 \le i \le 7$.

In the following, only the derivations for (c1) and (c5) are given. A derivation of (c4) is similar to (c5) while the remaining derivations are similar to that of (c1).

2.1.1. A derivation of (c1)

The first sequent in the following derivation is obtained by thinning from the last sequent in section 5.1, expressing that the strings <c1,c2,SeqCmd> and <e,c,WCmd> are provably distinct.

e:Exp, c:Cmd, c1:Cmd, c2:Cmd, <c1,c2,SeqCmd> = <e,c,WCmd> →

—

e:Exp, c:Cmd, c1:Cmd, c2:Cmd, r:S, t:S
→ *(<c1,c2,SeqCmd> =<e,c,WCmd> ⊃ <r,t>:W)
c1:Cmd, c2:Cmd, r:S, s:S, t:S, <c1,r,s>:CmdSem, <c2,s,t>:CmdSem
→ <<c1,c2,SeqCmd>,r,t>:CmdSem [From defn of CmdSem]

---

e:Exp, c:Cmd, c1:Cmd, c2:Cmd, r:S, s:S, t:S, W:WCls[e,c],
<c1,r,s>:CmdSem, (c1=<e,c,WCmd> ⊃ <r,s>:W)
<c2,s,t>:CmdSem, (c2=<e,c,WCmd> ⊃ <s,t>:W)
→ *(<<c1,c2,SeqCmd>,r,t>:CmdSem
∧ (<c1,c2,SeqCmd> =<e,c,WCmd> ⊃ <r,t>:W))
e:Exp, c:Cmd, c1:Cmd, c2:Cmd, r:S, s:S, t:S, W:WCls[e,c],
*<c1,r,s>:T, *<c2,s,t>:T → *<<c1,c2,SeqCmd>,r,t>:T
e:Exp, c:Cmd, W:WCls[e,c] →
[∀c1,c2:Cmd][∀r,s,t:S]*(<c1,r,s>:T ∧ <c2,s,t>:T ⊃ <<c1,c2,SeqCmd>,r,t>:T)

End of derivation of (c1)

2.1.2. A derivation of (c5).

The sequent (c5) can be derived from from sequents (i) and (ii) as following.

(i) e:Exp, c:Cmd, e1:Exp, c1:Cmd,
W:WCls[e,c], <e1,c1,WCmd> = <e,c,WCmd>
→ W:WCls[e1,c1]

(ii) e1:Exp, c1:Cmd, r:S, s:S, t:S, *W:WCls[e1,c1],
<e1,r,1>:ExpSem, <c1,r,s>:CmdSem,
<s,t>:W → <r,t>:W

---

cut on W:WCls[e1,c1]

e:Exp, c:Cmd, e1:Exp, c1:Cmd, r:S, s:S, t:S, W:WCls[e,c], <e1,r,1>:ExpSem,
<c1,r,s>:CmdSem, <e1,c1,WCmd> = <e,c,WCmd>,
<s,t>:W → <r,t>:W
<e1,c1,WCmd> =<e,c,WCmd> → <e1,c1,WCmd> =<e,c,WCmd>

---

e:Exp, c:Cmd, e1:Exp, c1:Cmd, r:S, s:S, t:S, W:WCls[e,c], <e1,r,1>:ExpSem,
<c1,r,s>:CmdSem, *(<e1,c1,WCmd> =<e,c,WCmd> ⊃ <s,t>:W),
<e1,c1,WCmd> =<e,c,WCmd> → <r,t>:W

—

e:Exp, c:Cmd, e1:Exp, c1:Cmd, r:S, s:S, t:S, W:WCls[e,c], <e1,r,1>:ExpSem,

<c1,r,s>:CmdSem, (<e1,c1,WCmd> =<e,c,WCmd> ⊃ <s,t>:W)
→ * (<e1,c1,WCmd> =<e,c,WCmd> ⊃ <r,t>:W)
e1:Exp, c1:Cmd, r:S, s:S, t:S,
<e1,r,1>:ExpSem, <c1,r,s>:CmdSem, <<e1,c1,WCmd>,s,t>:CmdSem
→ <<e1,c1,WCmd>,r,t>:CmdSem

---

e:Exp, c:Cmd, e1:Exp, c1:Cmd, r:S, s:S, t:S, W:WCls[e,c], <e1,r,1>:ExpSem,
<c1,r,s>:CmdSem, (c1=<e,c,WCmd> ⊃ <r,s>:W),
<<e1,c1,WCmd>,s,t>:CmdSem, (<e1,c1,WCmd> =<e,c,WCmd> ⊃ <s,t>:W),
→ *(<<e1,c1,WCmd>,r,t>:CmdSem ∧ (<e1,c1,WCmd> =<e,c,WCmd>
⊃ <r,t>:W))
e:Exp, c:Cmd, e1:Exp, c1:Cmd, r:S, s:S, t:S, W:WCls[e,c],
<e1,r,1>:ExpSem, *<c1,r,s>:T , *<<e1,c1,WCmd>,s,t>:T
→ *<<e1,c1,WCmd>,r,t>:T
e:Exp, c:Cmd, W:WCls[e,c] →
[∀e:Exp][∀c:Cmd]*[∀r,s,t:S]*(<e,r,1>:ExpSem ∧ <c,r,s>:T
⊃ (<<e,c,WCmd>,s,t>:T ⊃ <<e,c,WCmd>,r,t>:T)

2.1.2.1. A derivation of (i).

The rules =→ and <>→, derived in [Gilmore89], are used in the derivation of (i). The first rule is

$$\frac{\Gamma \to \Theta, [ta/u]F \qquad [tb/u]F, \Delta \to \Lambda}{\Gamma, \Delta, ta{=}tb \to \Theta, \Lambda}$$

and the second is one of the following three rules

$$\frac{\Gamma, ta1{=}tb1 \to \Theta}{\Gamma, <ta1,ta2,ta3>{=}<tb1,tb2,tb3> \to \Theta} \qquad \frac{\Gamma, ta2{=}tb2 \to \Theta}{\Gamma, <ta1,ta2,ta3>{=}<tb1,tb2,tb3> \to \Theta}$$

$$\frac{\Gamma, ta3{=}tb3 \to \Theta}{\Gamma, <ta1,ta2,ta3>{=}<tb1,tb2,tb3> \to \Theta}$$

where **F** is any formula in which only the variable u has a free occurrence, **ta**, **tb** are any closed terms and **ta1**, **tb1**, **ta2**, **tb2**, **ta3**, **tb3** any closed first order terms.

A derivation of (i) follows:

c:Cmd, e1:Exp, W:WCls[e1,c]

→ W:WCls[e1,c]
e1:Exp, c1:Cmd, W:WCls[e1,c1]
→ W:WCls[e1,c1]

=→

c:Cmd, e1:Exp, c1:Cmd, W:WCls[e1,c], *c=c1
→ W:WCls[e1,c1]
e:Exp, c:Cmd, W:WCls[e,c]
→ W:WCls[e,c]

=→

e:Exp, c:Cmd, e1:Exp, c1:Cmd,
W:WCls[e,c], *e=e1, c=c1
→ W:WCls[e1,c1] <>→
e:Exp, c:Cmd, e1:Exp, c1:Cmd,
W:WCls[e,c], e1 = e, *<e1,c1,WCmd> = <e,c,WCmd>
→ W:WCls[e1,c1] <>→ , contraction
e:Exp, c:Cmd, e1:Exp, c1:Cmd,
W:WCls[e,c], *<e1,c1,WCmd> = <e,c,WCmd>
→ W:WCls[e1,c1]

2.1.2.1. A derivation of (ii).

e1:Exp, r:S, <e1,r,1>:ExpSem → <e1,r,1>:ExpSem
c1:Cmd, r:S, s:S, <c1,r,s>:CmdSem → <c1,r,s>:CmdSem
<s,t>:W → <s,t>:W
<r,t>:W → <r,t>:W

---

e1:Exp, c1:Cmd, r:S, s:S, t:S,
* (<e1,r,1>:ExpSem ∧ <c1,r,s>:CmdSem ∧ <s,t>:W ⊃ <r,t>:W),
<e1,r,1>:ExpSem, <c1,r,s>:CmdSem,
<s,t>:W → <r,t>:W
r:S → r:S
s:S → s:S
t:S → t:S

---

e1:Exp, c1:Cmd, r:S, s:S, t:S,
* [∀r,s,t:S] (<e1,r,1>:ExpSem ∧ <c1,r,s>:CmdSem ∧ <s,t>:W ⊃ <r,t>:W),
<e1,r,1>:ExpSem, <c1,r,s>:CmdSem,
<s,t>:W → <r,t>:W

e1:Exp, c1:Cmd, r:S, s:S, t:S, *W:WCls[e1,c1],
<e1,r,1>:ExpSem, <c1,r,s>:CmdSem,
<s,t>:W → <r,t>:W

End of derivation of (c5)

End of proof of lemma 5.4.1