MAT.ES 2005

Primer Congreso Conjunto de Matemáticas RSME-SCM-SEIO-SEMA

Valencia, 31 enero - 4 febrero

Programa de Sesión Especial (N^o. 14):

Lógica en la Computación

Organizador: Argimiro Arratia

Expositores

(1) José Luis Balcázar (Universitat Politécnica de Cataluyna)
(2) María Luisa Bonet (Universitat Politécnica de Cataluyna)
(3) Victor Dalmau (Universitat Pompeu Fabra)
(4) Joerg Flum (Universität Freiburg)
(5) Félix F. Lara Martín (Universidad de Sevilla)
(6) Francisco López Fraguas (Universidad Complutense de Madrid)
(7) Elvira Mayordomo (Universidad de Zaragoza)
(8) Luis Miguel Pardo (Universidad de Cantabria)
(9) Argimiro Arratia (Universidad de Valladolid)

Programa sugerido

Esta Sesión se reune Jueves 3 de Febrero, 15:30-17:30 y 18:00-20:00 en Aula 10; y Viernes 4 de Febrero, 11:30-13:30 en Aula 7.

Jueves 3

(15:30): Luis Miguel Pardo
(16:10): José Luis Balcázar
(16:50): Francisco López Fraguas
(17:30): Descanso
(18:00): Elvira Mayordomo
(18:40): Victor Dalmau
(19:20): Joerg Flum
(20:00): Fin

Viernes 4

(11:30): María Luisa Bonet
(12:10): Félix F. Lara Martín
(12:50): Argimiro Arratia
(13:30): Fin

Títulos y resúmenes de exposiciones

Extensiones de cláusulas de Horn: Aprendizaje y propiedades estructurales.

José Luis Balcázar
LARCA Lab., LSI Dept,
Universitat Politécnica de Cataluyna,
balqui@lsi.upc.es

Esta presentación explora los límites actualmente conocidos de la aprendibilidad bajo preguntas de fórmulas booleanas restringidas. Es sabido que es posible aprender bajo preguntas teorías Horn en tiempo cuadrático. Aqui aportamos ante todo una reformulación esencialmente estructural de esta propiedad, que explica la relación entre la aprendibilidad y la propiedad de cierre bajo intersección, que a su vez conlleva una conexión bien definida con operadores de clausura. Exploramos la utilidad de este nuevo enfoque para extender los resultados, obteniendo un nuevo algoritmo de aprendizaje, así como una caracterización estructural, similar al cierre bajo intersección, para teorías axiomatizables por conjunciones de cláusulas de Horn y cláusulas de Sagiv. Enumeraremos las distintas lineas de posible progreso que se pueden seguir de este trabajo, en busca de algoritmos óptimos para aprendizaje de cláusulas de Horn o de fórmulas más generales que éstas.

Automatizability of Propositional Proof Systems.

María Luisa Bonet
Universitat Politécnica de Cataluyna, Barcelona
bonet@lsi.upc.es

En la charla nos plantearemos dos preguntas:

- Cuán grande es (en tamaño) la menor demostración de una tautología, en un sistema de demostración dado?
- Cuán costoso es (en tiempo) encontrar esta demostración?

La primera de las preguntas, ya fue planteada por Cook y Reckhow hace tiempo. Su motivación era más bien teórica, ya que está relacionada con la igualdad de NP y $CoNP$, y por tanto con la igualdad de P y NP, pero llevó a muchos investigadores a estudiar los diferentes sistemas de demostración proposicionales existentes desde un punto de vista de complejidad, y a jerarquizarlos según su potencia.

La segunda pregunta, bastante más práctica, es la que nos planteamos analizar en esta charla.

Para contestar a la primera pregunta, Cook y Reckhow definieron los **sistemas de demostración proposicionales acotados polinómicamente** como aquellos para los cuales existía un polinomio $p(x)$ tal que, para toda tautología x de tamaño $|x|$, puede encontrarse una demostración y de tamaño $|y|$ que satisface $|y| \leq p(|x|)$. Hoy sabemos que muchos de los sistemas de demostración proposicional conocidos no están acotados polinomicamente, y la pregunta se mantiene abierta para el sistema de demostración de Frege y sus extensiones.

Para contestar a la segunda pregunta, definimos los **sistemas automatizables** como aquellos para los que existe un algoritmo que permite encontrar una demostración de cualquier tautología en tiempo polinomico en el tamaño de la menor demostración de la tautología.

Iniciamos así un ambicioso proyecto de investigación en que nos proponiamos encontrar algún sistema de demostración que fuese automatizable en este sentido. Hablaremos de que sistemas de demostración no son automatizables (bajo supuestos muy fuertes), y de que sistemas de demostración tienen algorítmos de búsqueda de demostraciones más o menos eficientes. También mencionaremos los sistemas de demostración sobre los cuales no sabemos nada al respecto.

Sobre la robustez de los "existential pebble games"

Victor Dalmau
Universitat Pompeu Fabra, Barcelona
victor.dalmau@upf.edu

Los "existential pebble games"fueron definidos originalmente con el objetivo de ser utilizados como una herramienta para el estudio de la expresividad de "Datalog", una lógica ampliamente usada en teoría de bases de datos.

En esta charla revisaremos algunas aplicaciones recientes de los "existential pebble games"en diversos campos de la teoría de la complejidad computacional, tales como complejidad de la demostración y la satisfacción de restricciones.

El teorema de Fagin y complejidad paramétrica

Joerg Flum
Universität Freiburg, Alemania
joerg.flum@math.uni-freiburg.de

El teorema de Fagin caracteriza los problemas en **NP** mediante las fórmulas Σ_1^1 de la lógica de segundo orden. Este teorema demostrado

hace 30 años fue el punto de partida para la caracterización de las clases de complejidad clásica.

En la charla haremos hincapié en otro aspecto del teorema de Fagin: veremos que permite un estudio más refinado de la complejidad tanto clásica como paramétrica de varios problemas. A la par introduciremos las nociones básicas de complejidad paramétrica.

Funciones primitivas recursivas demostrablemente totales

A. Cordón Franco, A. Fernández Margarit y F. F. Lara Martín

Depto. de Ciencias de la Computación e Inteligencia Artificial. Facultad de Matemáticas (Universidad de Sevilla) C/Tarfia s/n, 41012 Sevilla.

Un álgebra de funciones es una familia de funciones que puede describirse como la menor clase que contiene algunas funciones básicas y es cerrada bajo ciertos operadores. Entre los ejemplos clásicos tenemos la clase de las funciones primitivas recursivas, $\mathcal{PR}$, y las clases $\mathcal{E}^n$ $(n \geq 0)$ de la jerarquía de Grzegorczyk. Otro ejemplo importante es $\mathcal{R}(\mathbf{T})$, la clase de la funciones recursivas demostrablemente totales (f.r.d.t) de una teoría $\mathbf{T}$ en el lenguaje de la Aritmética de primer orden. La clase $\mathcal{R}(\mathbf{T})$ puede usarse para obtener resultados de independencia acerca de $\mathbf{T}$ y para separarla de otras teorías. Además, si un álgebra de funciones, $\mathcal{C}$, es la clase de f.r.d.t. de una teoría $\mathbf{T}$, entonces pueden utilizarse los métodos de la Teoría de la Demostración y la Teoría de Modelos aplicables a $\mathbf{T}$ para establecer resultados sobre $\mathcal{C}$, incrementando así los métodos disponibles en el estudio de las álgebras de funciones.

Por otra parte, mediante álgebras de funciones pueden obtenerse caracterizaciones "independientes de máquina" de muchas clases de complejidad (ver [1]), ofreciendo una versión alternativa de algunos problemas importantes de la Teoría de la Complejidad. De este modo las clases de funciones recursivas demostrablemente totales constituyen un punto de conexión entre la Teoría de la Complejidad, la Teoría de la Demostración y la Teoría de Modelos.

En [2] presentamos un nuevo ejemplo de las interacciones entre fragmentos de la Aritmética, álgebras de funciones y complejidad computacional. Para cada álgebra de funciones, $\mathcal{C}$, se define el álgebra $\mathcal{E}^{\mathcal{C}}$ como la menor clase que contiene las funciones básicas (cero, sucesor y proyecciones) y es cerrada bajo composición y recursión $\mathcal{C}$–acotada.

Estudiamos las relaciones entre $\mathcal{C}$ y $\mathcal{E}^{\mathcal{C}}$ en el caso en que $\mathcal{C}$ es la clase de f.r.d.t. de alguna teoría $\mathbf{T}$. Si $\mathcal{C} = \mathcal{R}(\mathbf{T})$, entonces

(1) $\mathcal{C} \cap \mathcal{PR} \subseteq \mathcal{E}^{\mathcal{C}}$.
(2) Si, además, $\mathcal{C}$ es cerrada bajo minimización acotada, entonces
   (a) $\mathcal{E}^{\mathcal{C}}$ es el cierre de $\mathcal{C} \cap \mathcal{PR}$ bajo composición y recursión acotada.
   (b) $\mathcal{C} \cap \mathcal{PR} = \mathcal{E}^{\mathcal{C}}$ si y sólo si existe una teoría $\mathbf{T}'$ tal que $\mathcal{E}^{\mathcal{C}} = \mathcal{R}(\mathbf{T}')$.

Para la prueba de estos resultados se introduce la noción de álgebra de funciones Δ_0–generada. Un álgebra de funciones, $\mathcal{C}$, es Δ_0–generada si contiene la clase de Grzegorcyk $\mathcal{M}^2$ y cada función de $\mathcal{C}$ puede expresarse como la composición de dos funciones de $\mathcal{C}$ cuyo grafo es Δ_0–definible. Se tiene que:

- Un álgebra de funciones es Δ_0–generada si y sólo si es la clase $\mathcal{R}(\mathbf{T})$ de alguna teoría $\mathbf{T}$ (extensión de $\mathbf{I}\Delta_0$).

En consecuencia, si $\mathcal{C} \subseteq \mathcal{PR}$, entonces, por (2), $\mathcal{C} = \mathcal{E}^{\mathcal{C}}$ si y sólo si $\mathcal{E}^{\mathcal{C}}$ es Δ_0–generada. Este hecho tiene interesantes aplicaciones para las clases de complejidad $\mathcal{F}_{\mathrm{PH}}$ (funciones computables en la jeraquía de tiempo polinomial) y $\mathcal{F}_{\mathrm{LTH}}$ (funciones computables en la jerarquía de tiempo lineal). Ambas clases están contenidads en $\mathcal{PR}$ y son Δ_0–generadas obteniéndose:

- Si $\mathcal{F}_{\mathrm{PSPACE}}$ es Δ_0–generada, entonces $\mathcal{F}_{\mathrm{PSPACE}} = \mathcal{F}_{\mathrm{PH}}$.
- Si $\mathcal{F}_{\mathrm{LINSPACE}}$ is Δ_0–generada. entonces $\mathcal{F}_{\mathrm{LINSPACE}} = \mathcal{F}_{\mathrm{LTH}}$. Equivalentemente, $\mathcal{E}^2 = \mathcal{M}^2$ si y sólo si $\mathcal{E}^2$ is Δ_0–generada.

Estos hechos sugieren que un conocimiento más profundo de las propiedades estructurales de las álgebras Δ_0–generadas (así como la construcción de álgebras de funciones que no son Δ_0–generadas) podría ser relevante en el estudio de estas clases de complejidad. Por otro lado también plantean la siguiente pregunta: Si $\mathcal{C} = \mathcal{R}(\mathbf{T})$ y $\mathcal{E}^{\mathcal{C}}$ es Δ_0–generada, ¿existe alguna teoría natural, $\mathbf{T}'$, tal que $\mathcal{R}(\mathbf{T}') = \mathcal{E}^{\mathcal{C}}$?

El estudio de los esquemas de inducción restringidos a fórmulas Δ_1 nos proporciona una respuesta. Sea $\mathbf{I}\Delta_1(\mathbf{T})^-$ la teoría axiomatizada por el esquema de inducción restringido a fórmulas $\Delta_1(\mathbf{T})$ (sin parámetros). Entonces

- $\mathcal{R}(\mathbf{I}\Delta_1(\mathbf{T})^-) = \mathcal{C} \cap \mathcal{PR}$.

Estos resultados se obtienen a partir de teoremas de conservación entre fragmentos de la Aritmética, que probamos mediante técnicas de la Teoría de Modelos.

Referencias

[1] Clote, P. *Computation Models and Function Algebras*. En Handbook of Computability Theory, pgs. 589–681. North-Holland, 1999.

[2] Cordón Franco A.; Fernández Margarit, A. y Lara Martín, F. F. *Provably Total Primitive Recursive Functions: Theories with Induction*. En Computer Science Logic: 18th International Conference, CSL 2004, Karpacz, Polonia. Lecture Notes in Computer Science, Vol. 3210, pgs. 355–369, Springer–Verlag, 2004.

Bases lógicas para la verificación de programas lógico-funcionales [1]

J.M Cleva Millor, J. Leach Albert, F. Javier López Fraguas

Departamento de Sistemas Informáticos y Programación, UCM, Madrid

{jcleva,leach,fraguas}@sip.ucm.es

Programación declarativa Bajo la denominación genérica de *programación declarativa* se engloban aquellos estilos de programación en los que la lógica del programa está cerca de la lógica del problema a resolver, en lugar de estar cerca de la lógica de la máquina, como sucede en la programación imperativa. Es más programación por *descripción* que por *prescripción*. Los programas declarativos se ajustan a la idea de *especificación ejecutable*.

Los dos paradigmas fundamentales de la programación declarativa —la programación funcional y la programación lógica— admiten fundamentarse desde un punto de vista lógico. La programación funcional, en la que se definen funciones mediante ecuaciones (sistemas de reescritura), puede basarse en la lógica ecuacional, mientras que la programación lógica, de estilo axiomático, se basaría en la lógica de primer orden, más exactamente en un fragmento suyo, la lógica de Horn. En ambos casos, los programas se pueden entender como teorías en la lógica en cuestión y los cómputos se corresponden con deducciones.

Programación declarativa multiparadigma Desde hace un par de décadas se está dedicando un esfuerzo considerable a diseñar lenguajes que aúnen las mejores características de la programación lógica y la programación funcional. Los lenguajes resultantes se denominan lenguajes lógico-funcionales (ver [4] para una panorámica ya clásica). En casi todos ellos los programas son algún tipo de sistemas de reescritura y el mecanismo de cómputo es alguna variante del *estrechamiento* ('*narrowing*', ver e.g. [4]), que combina la reescritura de la programación

[1] Trabajo parcialmente financiado por el proyecto MELODIAS (TIC2002-01167).

funcional con la unificación de la programación lógica. Suele seguirse además la *disciplina de constructoras*, que separa el repertorio de símbolos sintácticos de función en dos conjuntos disjuntos: el de las constructoras de datos, que sirven para formar expresiones no evaluables (c-términos) que determinan un universo de valores o datos, y el de las funciones propiamente dichas, a definir mediante reglas en el programa, que pueden formar expresiones evaluables.

CRWL: Un marco lógico para la programación lógico-funcional Un rasgo distintivo de los sistemas lógico-funcionales modernos es que aceptan como programas sistemas de reescritura no terminantes y no confluentes, que definen por tanto funciones no estrictas y no deterministas. Una consecuencia de ello es que la lógica ecuacional no es válida para interpretar los programas. Por ejemplo, en un programa que conste de las reglas $f(X) \to X$ y $f(X) \to X + 1$ no podemos entender las reglas como ecuaciones, pues eso nos conduciría a que, por ejemplo, $f(0) = 0$ y también $f(0) = 1$, y por tanto $0 = 1$, que no es lo pretendido.

Por ello, en [3] se propone el marco lógico CRWL que es comúnmente aceptado como un fundamento semántico adecuado para tal tipo de programas. El marco CRWL incluye:

• Un cálculo de pruebas para probar sentencias de la forma $e \to t$, expresando que la expresión e puede reducirse al c-término t.

• Una teoría de modelos, cumpliéndose que todo programa tiene un modelo distinguido (modelo inicial) con la propiedad de que la deducibilidad en el cálculo se corresponde con validez en todos los modelos y a su vez con validez en el modelo inicial.

• Un cálculo de estrechamiento como mecanismo de cómputo, correcto y completo respecto al cálculo de pruebas.

El marco CRWL, implementado en el sistema *Toy*, se ha extendido para cubrir muchos otros aspectos de la programación declarativa: orden superior, tipos polimórficos y algebraicos, géneros ordenados, restricciones genéricas y sobre distintos dominios concretos, orientación a objetos, negación constructiva (véase [7] para una panorámica).

Demostración de propiedades de CRWL-programas Un aspecto importante dentro de la producción de software es la verificación, la comprobación de que los programas satisfacen ciertos requisitos de especificación. Los métodos formales, en particular los basados en la lógica, son herramientas adecuadas para estos propósitos.

Aquí nos interesa desarrollar métodos para probar propiedades de CRWL-programas. Las propiedades de interés son aquellas fórmulas de primer orden sobre la relación de reducción $e \to t$ y un predicado

$cterm(x)$ (que expresa la propiedad de ser c-término) que expresen propiedades válidas en el modelo inicial del programa.

Las ideas fundamentales del enfoque que seguimos, iniciado en [2], se basan en trasladar CRWL-programas a programas lógicos, lo que en definitiva supone trasladar la lógica CRWL a la lógica de primer orden, mucho mejor conocida. Más en concreto, estas ideas se plasman en los siguientes pasos:

• Dado un $CRWL$-programa P, se construye un programa lógico asociado (i.e. una teoría de Horn) P_L de modo que el modelo mínimo de Herbrand de P_L y su teoría T_{P_L} se correspondan bien con el CRWL-modelo inicial de P y su teoría T_P.

• Típicamente, por argumentos a lo Gödel, la teoría T_{P_L} no es recursivamente enumerable, y solo podemos dar métodos que la aproximen. Una cadena de aproximaciones en que cada una refina a la anterior viene dada por las teorías (de primer orden) de P_L, de su compleción de Clark y de la extensión inductiva de la compleción.

• Una forma interesante de mejorar aún la aproximación a T_{P_L} resulta de incorporar explícitamente las CRWL-derivaciones a la axiomatización en primer orden de la relación $e \rightarrow t$. De este modo pueden probarse propiedades que involucran razonamientos sobre no terminación.

Estas ideas se han llevado a la práctica en varios asistentes interactivos como *Isabelle* [6], *LPTP* [8] o *ITP* [1].

Referencias

[1] M. Clavel. *The ITP tool*. Proc. of the 1st Workshop on Logic and Language, 55–62, 2001.

[2] J.M. Cleva, J. Leach, F.J.López-Fraguas. A logic programming approach to the verification of functional-logic programs. Proc. Principles and Practice of Declarative Programming (PPDP'04), ACM Press, pp. 9-19, 2004.

[3] J.C. González-Moreno, M.T. Hortalá-González, F.J. López-Fraguas, M. Rodríguez-Artalejo. *An Approach to Declarative Programming Based on a Rewriting Logic*. Journal of Logic Programming 40(1), pp. 47–87, 1999.

[4] M. Hanus. *The Integration of Functions into Logic Programming: From Theory to Practice*. Journal of Logic Programming 19&20, pp. 583–628, 1994.

[5] F.J. López Fraguas, J. Sánchez Hernández. *$\mathcal{TOY}$: A Multiparadigm Declarative System*. Proc. RTA'99, Springer LNCS 1631, pp 244–247, 1999.

[6] T. Nipkow, L.C. Paulson, M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higer-Order Logic*. Springer LNCS 2283, 2002.

[7] M. Rodríguez-Artalejo. *Functional and Constraint Logic Programming*. in H. Comon, C. Marché and R. Treinen (eds.), *Constraints in Computational Logics, Theory and Applications*, Revised Lectures of the International Summer School CCL'99, Springer LNCS 2002, Chapter 5, pp. 202–270, 2001.

[8] R.F. Stäerk. *The theoretical foundations of LPTP (A logic program theorem prover)*. Journal of Logic Programming 36, pp. 241–269, 1998.

La dimensión de Hausdorff efectiva y la compresión de secuencias binarias.

Elvira Mayordomo

Universidad de Zaragoza, Dept. de Informática e Ingeniería de Sistemas, Edificio Ada Byron, Mara de Luna 1 - E-50018, Zaragoza,

elvira@unizar.es

La dimensión de Hausdorff en el espacio de secuencias infinitas de Cantor admite caracterizaciones por medio de estrategias de juego o "galas"(Lutz 2000) y por medio de algoritmos de predicción (Hitchcock 2003).

Estas caracterizaciones permiten plantear de manera natural versiones constructivas y efectivas de dimensión. Además, algunas de estas versiones admiten caracterizaciones por medio de complejidad de Kolmogorov o cantidad de información. En concreto esto es cierto para las versiones constructiva y acotada en espacio, por ejemplo la dimensión "calculada con espacio polinómico" coincide con el grado de compresión alcanzable con cotas de espacio polinómicas.

En complejidad computacional el recurso más valorado es el tiempo, y el tipo de algoritmos de compresión más estudiados son aquellos que trabajan en tiempo polinómico. Por otro lado, una caracterización que utilice complejidad de Kolmogorov acotada en tiempo no parece viable. Recientemente hemos encontrado una caracterización de dimensión "en tiempo polinómico" utilizando un tipo concreto de compresores reversibles que trabajan en tiempo polinómico.

How upper and Lower Complexity Bounds meet in Elimination Theory

Luis Miguel Pardo

Departamento de Matemáticas, Estadística y Computación
Facultad de Ciencias, Universidad de Cantabria
Avda. Los Castros s/n, E-39071 SANTANDER

luis.pardo@unican.es

The central problem in efficient elimination theory is *Hilbert's Nullstellensatz*. This problem can be stated in the following terms:
Given multivariate polynomials $f_1, \ldots, f_s \in \mathbb{Q}[X_1, \ldots, X_n]$ *decide whether they share a common zero. Namely, decide whether*

$$\exists x_1 \in \mathbb{C}, \ldots, \exists x_n \in \mathbb{C}, f_i(x_1, \ldots, x_n) = 0, 1 \leq i \leq s.$$

All **NP**–complete problems may be rewritten as a particular instance of this one. Valiant's $\sharp$**P**–completeness can also be formulated by adding counting number solutions as question. However, the problem contains some particular features difficult to fix. In fact, the problem decomposes in two subproblems: *the Consistency Problem* and the *Solving Problem*. The Consistency Problem is the problem of selecting the appropriate candidate to test (much as in Cook's Conjecture) whereas the Solving Problem consists on preparing the data to be selected to answer questions. These two problems are part of the list of problems stated by S. Smale for the 21st century. This talk is mainly concerned with a survey on the state of the art of both problems, including algebraic–geometric, numerical and diophantine techniques.

Esquemas de Programas

Argimiro Arratia
Depto. de Matemática Aplicada,
Facultad de Ciencias, Universidad de Valladolid
arratia@mac.uva.es

Hablaré sobre una manera formal de definir clases de problemas que es mas 'computacional'que emplear fórmulas lógicas (como en la Teoría Descriptiva de la Complejidad) y más parecido a lo que entendemos por un lenguaje de programacion que una máquina de Turing. Estos son los esquemas de programas o *program schemes*.

Una forma de esquemas de programas se basa en bucles tipo WHILE, además de instrucciones de asignación y (posiblemente) evaluación nodeterminísitica (mediante instrucción GUESS(x) que asigna nodeterminísticamente un valor a la variable x entre los posibles valores del universo finito de la estructura de datos de entrada). Estos programas constituyen la clase **NPS** y, esencialmente, modelan la programación secuencial, lo cual se evidencia cuando imponemos un orden implícito en el universo de los datos de entrada, para tener la clase **NPS**(suc), y se demuestra que esta clase coincide con la clase de problemas aceptados por máquinas de Turing nodeterminísticas que utilizan espacio

logaritmico: **NPS**(suc) = **NLOG**. Más aún, si agregamos a los esquemas de programas en **NPS** los siguientes medios de manipulación de datos: (1) pilas, y (2) arreglos; obtenemos, respectivamente, las clases **NPSS** y **NPSA**, y si imponemos el orden se tienen las respectivas clases de programas con orden **NPSS**(suc) y **NPSA**(suc) y se tiene: **NPSS**(suc) = **P** y **NPSA**(suc) = **PESPACIO**.

Cosas interesantes suceden al liberarnos del orden, tales como la existencia de jerarquías de clases de programas, que se establecen con técnicas importadas de la Teoría de Modelos Finitos; además se pueden traducir al formalismo de los esquemas de programas algunos de los problemas abiertos importantes de la teoría de la complejidad computacional como: ¿es **NP** = **PESPACIO**?

Actualmente trabajamos con otra forma de esquemas de programas basados en bucles tipo FORALL, los cuales constituyen un modelo natural de programación en paralelo, subceptibles de manipulación y fácil análisis como los programas en **NPS**.