The 6th International Conference on Software Engineering and Knowledge Engineering
A Pipeline CASE tool for Database Design

Naphtali Rishe and Wei Sun

School of Computer Science, Florida International University, University Park, Miami, FL 33199

Abstract. We have developed a tool for design of relational databases, including schemas, integrity constraints, reports, and data entry forms, using semantic binary schemas. The tool is based on a top-down methodology. In this methodology, a conceptual description of an enterprise is designed using a semantic binary model. Then, this description is converted into the relational database design. The tool automates virtually all the busy work of design. With respect to the intelligent design decisions, the tool accepts instructions from its user, who is a database designer, or, when the user defaults, makes decisions itself based on "rule-of-thumb" principles guided by the knowledge of the database's semantics. The tool creates a turn-key database application and its documentation with graphically-illustrated design reports, manuals, application glossaries, and data dictionaries, as well as an application-customized report generator. Changes in the semantic description or designer's instructions are propagated into the products.

1. INTRODUCTION

In the database design methodology automated by our tool, semantic binary schemas are converted into relational schemas and integrity constraints. The semantic database models offer a simple, natural, implementation-independent, flexible, and non-redundant specification of information and its semantic aspects. Since the original idea of [Abrial-74], many semantic data models have been studied in the Computer Science literature. Many semantic models have been surveyed in [Hull&King-87] and [Peckham&Maryanski-88]. Although somewhat differing in their terminology and their selection of tools used to describe the semantics of the real world, the various semantic models are roughly equivalent. This paper's methodology uses the Semantic Binary Model (SBM) ([Rishe-92-DDS], [Rishe-89-SD]) a descendant of the model of [Abrial-74]. SBM does not have as rich an arsenal of tools for semantic description as can be found in some other semantic models.

Nevertheless, the SBM has a small set of sufficient simple tools by which all of the semantic descriptors of the other models can be constructed. The use of semantic models for the design of relational schemas has been studied in [Brodie&al.-84-CM], [Chen-76], [King&McLeod-85], [Shoval-85], [Teorey&al.-86], [Leung&Nijssen-87], [Shoval&Even-Chaime-87], [Verheijen&VanBekkum-82], [Rosenthal&Reiner-90], [DeTroyer&Meersman-86], and other works. A graphical interactive system for the design of semantic databases is discussed in [Shoval&al.-88].

Our database design methodology (see also [Rishe-92-DDS]) differs in satisfaction of a broad range of schema-quality criteria, comparative analysis of different design choices in various steps of design, and systematic generation of integrity constraints. The methodology employs procedures for generation of keys of categories and for partitioning of non-disjoint categories into disjoint ones. The treatment of sub-super categories and non-disjoint categories is important for the proper reflection of the original semantics in the resultant relational schema and for avoidance of logical redundancy of information in the database. In the input semantic model of this methodology, SBM, the semantic issues that are rather simple to the user are graphically explicit. Other semantic nuances are relegated to integrity constraints, and they are propagated into the relational schema's external integrity constraints. After a brief description of our database design methodology, this paper introduce a CASE tool for automatic database design.

2. DATABASE SCHEMAS: SEMANTIC AND RELATIONAL

2.1. The Semantic Binary Model

This section describes the Semantic Binary Model. A more detailed description can be found in [Rishe-92-DDS]. The semantic binary database model represents information of an application's world as a collection of elementary facts of two types: unary facts categorizing objects of the real world and binary facts establishing relationships of various kinds between pairs of objects. A definition of the model's concepts follows.
Object — any item in the real world. It can be either a concrete object or an abstract object as follows. 

Value, or Concrete Object — a printable object, such as a number, a character string, or a date. Abstract Object — a non-value object in the real world. An abstract object can be, for example, a tangible item (such as a person, a table, a country), or an event (such as an offering of a course by an instructor), or an idea (such as a course). Abstract objects cannot be represented directly in the computer.

Category (also called Entity Type or Entity Set in some semantic models) — any concept of the application’s real world which is used for classification of objects. Two categories are disjoint if no object may simultaneously be a member of both categories. A category is a subcategory of another category if at every point in time every object of the former category should also belong to the latter. Binary Relationship — any concept of the application’s real world which is a binary property of objects, that is, the meaning of a relationship or connection between two objects. Notation: “x R y” means that object x is related by the relationship R to object y. Binary relationships are classified as many-to-one (m:1, functional), one-to-many (1:m), many-to-many (m:m), and one-to-one (1:1). The descriptor proper may be used: e.g. proper m:1, means that the relation is m:1 and not 1:1.

A category C is the domain of R if it satisfies the following two conditions: (a) whenever xRy then x belongs to C (at every point in time for every pair of objects); and (b) no proper subcategory of C satisfies (a). A category C is the range of R if: (a) whenever xRy then y belongs to C (at every point in time for every pair of objects); and (b) no proper subcategory of C satisfies (a). A relationship R whose domain is C is total if at all times for every object x in C there exists an object y such that xRy. (At different times different objects y may be related to a given object x.)

A non-binary relationship is regarded in the Binary Model as group of several simple relationships, specifically:

(1) An abstract category of events. Each event symbolizes the existence of a relationship between a group of objects.

(2) Functional binary relationships, whose domain is the category (a). Each of those functional binary relationships corresponds to a role played by some objects in the non-binary relationship.

Thus, the fact that objects x1, ..., xn participate in an n-ary relationship R in roles R1, ..., Rn, is represented by: an object e in the category R' and binary relationships eR1x1, ..., eRnxn.

2.2. The Relational Model

For convenience of the database design and use of languages, this section defines the Relational Model technically as a subset of the Semantic Binary Model.

Attribute — a m:1 or 1:1 relationship whose range is a concrete category. Time-invariant attribute — An attribute A is time-invariant if once an object x becomes related by A to a value y, the object x will forever be related by A to y, as long as x exists. (There are no time-invariant attributes in the natural user world. Even if the laws of physics or society do not allow for an attribute to change in time, the attribute may change in the perceived real world due to discoveries of errors in earlier perception. For example, a social security number could be wrongly reported and then corrected. Thus, time-invariance is defined only in implementational restrictions. Such restrictions are unavoidable in the relational database design. The methodology of relational schema design that is presented below has among its goals the minimization of the negative effect of such implementational restrictions.)

A time invariant attribute of a category is called its [single-attribute] key if it is 1:1 and total. That means that the values of the attribute can be used to identify the objects of the category. (Due to the time-invariance requirement, no attribute is really a key in the natural user’s world. Thus, the property of a key is defined only in implementational restrictions, which are unavoidable in the relational database design. Also, the requirement of totality is very rarely an integrity constraint imposed by the logic of the user world, but rather is an implementational restriction.) Convention: In this paper, we shall name the attributes constrained to be keys with the suffix -key.

A [multi-attribute] key of a category C is a minimal collection of total time-invariant attributes f1, f2, ..., fn of category C such that: for any collection of values x1, ..., xn there is at most one object y in C such that x1 = y.f1 and x2 = y.f2 and ... and xn = y.fn

Convention: in this paper, when a category is constrained to have exactly one key, and the key is composed of several attributes, we shall name these attributes with the suffix -in-key.

A binary schema is called a relational schema if

(i) all the abstract categories of the schema have keys

(ii) all the abstract categories are pairwise disjoint

(iii) the only relationships are attributes.
2.3. Schema Design Goals

A schema is said to be of high quality if it satisfies the following criteria (described in greater detail in Chapter 1 of [Rishe-92-DDS] and Chapter 2 of [Batini&al.-92]): the schema is a natural description of the real world; contains very little or no redundancy; does not impose implementational restrictions; covers as many integrity constraints as possible; the schema is flexible to design changes; and other minor criteria of [Rishe-92-DDS].

The most important issue of the database design is the design of a high-quality schema within the restrictions of the available DBMS and database model. A low-quality schema increases the chances of corruption of the data, makes it very hard to use and maintain the database, and makes it very hard, if not impossible, to adjust the database to the changing concepts of the application’s real world. It is easy to design a high quality schema in semantic models, particularly the Semantic Binary Model. The task is much harder in the Relational Model. Moreover, it is usually impossible to describe an application world by a schema in the Relational Model with the same high quality as with which that application can be described in the Semantic Binary Model.

A schema-conversion is a replacement of a schema by another schema having the same information content. This means that each of the two schemas can be regarded as a user-view of the other. Schema-conversion is a means of database design: a schema is first designed in a higher-level database model and then translated into a lower-level model which is supported by the available DBMS (when a DBMS for a higher-level model is unavailable or inadequate).

3. SCHEMA CONVERSION: SBM TO RELATIONAL

The central part of our tool is the automation of the database design methodology of Chapter 3 of [Rishe-92-DDS]. That involves algorithms imitating the human designer: automatic restructuring of schemas, generation of names for new concepts; generation and propagation of integrity constraints, as well as choosing defaults for intelligent design decisions. This section describes the steps of that methodology with comments regarding their automation. The next section describes the other parts of the tool. In this paper, the constraints are specified in a form of first-order predicate calculus adapted to databases. A full description of this language is given in [Rishe&Sun-91-PC].

3.1. Composition and Split of Relationships

Two auxiliary definitions of terminology that will be used in the conversion algorithm follow.

Composition of relationships

Let the range of Relationship \( R_1 \) be the domain of Relationship \( R_2 \). Relationship \( R \) is the composition of \( R_1 \) and \( R_2 \) if:

for every \( x, y \): \( xRy \) if there exists \( z \) such that \( xR_1z \) and \( zR_2y \).

The composition of relations by an automatic tool involves creation of a new name, which can normally be composed of the old names. The designer can override such name generation by an instruction in the input. Also, a comment describing the new relation is automatically generated from the comments of the old relations.

Relationship-split — conversion of a schema having a relationship \( R \) into another schema having, instead of \( R \), a new abstract category \( C \) and two total functional relationships \( R_1, R_2 \), whose domain is \( C \), s.t. \( xRy \) if there exists an object \( z \) in \( C \) for which \( zR_1x \) and \( zR_2y \). In the process, the new category can be automatically given a name and a comment derived from the old relation and categories.

The following subsections present the conversion algorithm.

3.2. Keys

Step 1. Choose a key for every abstract category, excluding subcategories of other categories, as follows, in the order of preference:

1) single-attribute key — if the category has an attribute which is \( 1:1 \), time-invariant, and total;

2) "forced" single-attribute key — an attribute which can be implementationally restricted to be \( 1:1 \), time-invariant, and total;

3) multi-attribute key — a minimal collection of attributes which are time-invariant and total, and jointly identify all the objects in the category;

4) "forced" multi-attribute key — a minimal collection of attributes which can be implementationally restricted to be time-invariant and total, and to jointly identify all the objects of the category;

5) inferred key — a collection of attributes inferable from the information existing in the schema and from keys of other categories, such that these attributes can be implementationally restricted, to be time-invariant and total, and to jointly identify all the
objects of the category;

6) enumerator id key — create a new external randomized enumeration for the objects in the category.

When the database designer defaults, the latter id-key is created by our tool. The values of this attribute should bear no correlation to the other information in the database, since the other information may change in time, while the key is time-invariant. The data-entry forms generated by our tool automatically assign "meaningless" unique values to such attributes, thus relieving the end-user from the burden of creating such identifiers.

In the input of our system, the designer may specify that a given category has a semantic key: a set of relations and/or attributes that jointly identify the objects of the category. Our tool will find inferred keys of the categories by computing a transitive closure of such specifications (not all categories will have inferred keys). Assume that a category \( C \) is the domain of total functional relationships \( f_1, \ldots, f_n \) which jointly identify all the objects of the category. The above assumption means that there is an integrity constraint

\[
\forall x \in C: \forall y \in C: (x.f_1 = y.f_1 \land \cdots \land x.f_n = y.f_n) \Rightarrow x = y
\]

In this case, once the keys of the ranges of the functional relationships \( f_1, \ldots, f_n \) are known, a key of \( C \) can be inferred from them. Let the keys of the ranges be \( k_1, \ldots, k_n \). Let \( k_i\cdot k_i^\prime \) be the set of inferred attributes obtained by the composition of the attributes comprising the key \( k_i \) and the relationship \( f_i \). The key of \( C \) is contained in the union of compositions of the relationships \( f_i \) onto the keys of their ranges, that is,

\[
(\{k_1 \text{ of } f_1\}, \ldots, \{k_n \text{ of } f_n\})
\]

Notice that the key of \( C \) is contained in the above union of compositions. Usually the key of \( C \) is equal to that union of compositions, but sometimes it is properly contained.

### 3.3. Disjointness of Categories

**Step 2.** Convert the intersecting abstract categories into disjoint categories by one of the following procedures for every group of intersecting categories.

a. **Conversion into one category (Union)**

b. **Conversion into artificially disjoint categories of Hats**

c. **Conversion into Union+Hats**: we can retain the union category with its original relations and in addition have the hat categories to hold relations specific to the former subcategories.

Criteria to choose one of the above options are described in [Rishe-92-DDS].

### 3.4. Removal of Relationships

The steps of this section complete the process of schema conversion.

**Step 3.** Convert every proper 1:m or m:m relationship whose range is a concrete category into a new abstract category with its two functional relationships through a relationship-split.

**Step 4.** Convert every 1:m relationship into an m:1 relationship by changing its direction and its name.

**Step 5.** Convert every proper many-to-many relationship into a category and two functional relationships through a relationship-split.

**Step 6.** Choose a key for every category produced through a relationship-split as follows.

For every category which was obtained through a relationship-split, a key is contained in the union of the compositions of its two functional relationships on the keys of their ranges.

**Step 7.** Replace every m:1 relationship \( f \) whose range is an abstract category by the composition of \( f \) on the chosen key of its range, that is, by attributes \( b_1, \ldots, b_n \), where \( x.b_i = (x.f).a_i \) and \( a_1, \ldots, a_n \) is the chosen key of \( f \)'s range.

**Step 8.** Remove redundant non-key attributes.

**Step 9.** Translate the integrity constraints into the terms of the new schema.

### 4. STRUCTURE OF THE TOOL

This tool is based on pipeline database design principles: the semantic description of an enterprise is processed by a series of filters, changes in the semantic description are automatically propagated. The input consists of the listing of a linear description of the semantic schema, including the definitions of all the categories, relationships, and attributes, integrity constraints at the semantic level, designer's choices for the conversion decisions, and overwrites to be modified in the resulting relational schema. The input consists of sections, each forming a logical subschema. The subschemas are interconnected by common categories. The output of the tool consists of:

1. Logical design report. This report is independent of the DBMS to be used for the project.

   a. Graphical semantic subschemas and definitions of all of their concepts.
b. Summary of the semantic schema.
c. The relational schema and its integrity constraints.
d. Glossary, defining the meaning of all the application's attributes and tables.
e. Miscellaneous analysis.
f. A comprehensive index.

2. An ORACLE database, including:
   a. SQL definitions for all the tables, attributes, comments to the attributes (derived from the comments to the meaning in the semantic schema), keys, referential integrity specifications, specifications of checks to be performed on attribute values.
   b. Generation of screen data entry and update forms, including triggers to enforce integrity. For every table there are two forms: (i) a base form covering all the attributes of the table; (ii) as above, but also containing sub-windows for all the dependent tables connected to this table by 1:m relationships, i.e. the tables having referential integrity pointers to this table.

3. A system of smart data reports. On invocation the user is menu-prompted for the database table, an optional additional logically related table, optional data selection criteria, optional sort criteria (default: by the keys), etc. The tool evaluates the outer join of the two tables according to the logically related fields. (The outer join produces rows of the first table even if there are no matches in the second table. The system knows what fields to join on because it knows the semantics of the database.) Multi-row headers are added based on the names of the fields. Wide output is automatically compressed to fit into the minimal width by finding the widest actual datum in each column (or, when the datum or heading consists of several words which can be split into several lines, the widest word). If after compression the output still cannot fit in the width of one page, the output is split horizontally between separate pages, while repeating the values of the sort fields. For example, if the report consists of fields \( f_1, f_2, f_3, \ldots, f_{50} \), where \( f_1 \) and \( f_2 \) are the key fields, the output may appear in pages 1a, 1b, 1c, 2a, 2b, 2c, etc., where page 1a contains the fields \( f_1 \) to \( f_7 \), page 1b contains fields \( f_1, f_2, \) and \( f_8 \) to \( f_{40} \), and page 1c contains fields \( f_1, f_2, \) and \( f_{41} \) to \( f_{50} \).

This tool has been used for database design for the Everglades National Park.

Example.

A sample logical declaration of an attribute line in the input of the Everglades schema is:

```
attr location-tolerance
HYDROLOGY-STATION 0..1000
m:1 (Tolerance of the location of a station, in feet. A value \( X \) assigned to this attribute means that the tolerance is \( +/-X \) feet.)
```

Most of the above is a comment defining the meaning of the attribute. (This comment is automatically propagated to data-entry windows, reports, glossaries, etc.)

The input declarations are maintained in flat files using a text editor. Graphic depictions are automatically generated. Some other approaches prefer graphical input generated. In this tool, we prefer a textual input interface while leaving the pictures for automatic generation by the tool. This allows greater flexibility and saves time. About 80\% of the input is the text of the comments that are logical definitions of categories and relations. It is easier to maintain such comment texts using a text editor than using a graphic tool. Also, input hardware independence is achieved: any terminal and a modem will do.

Apart of the design pipeline, the input files are subjected to other tools like spellers, searchers, and publishing systems.

The input contains information based on interviews with the Client, translated into formal concepts. The specification of every concept consists of: the concept's name, which should be clear and meaningful to the database users; technical characteristics of the concept; and comment defining the meaning of the concept.

The purposes of the comment are:

- to verify that the systems analysts correctly understand the meanings of the application's concepts;
- to concisely convey the meanings of the application to the programming personnel who will work on the application in the future;
- to provide online comments on all database entities to the future users of the database on the Client's side: during the automatic schema design process the comments are propagated into data entry forms (as pop-up helps), data reports, etc. As new compound concepts are generated during the automatic design process, their comments are
compounded, derived and transformed (by simple syntactic manipulations).

- to provide an information reference manual for use by the Client's personnel and for training of new employees, whether they will be using the database or not;
- to facilitate decision making at the Client's managerial and executive levels by providing a graphic overview and a comprehensive directory of the information owned by the Client (as a supplement to the other decision support resources: a directory of the personnel employed, a directory of financial and tangible assets owned, and the database itself.)

Technical characteristics involve constraints on concepts. Also, a range of the possible values of an attribute can be given. For example, 23.5..100.7 means that the attribute is numeric, that its values may not be less than 23.5 or greater than 100.7, and that the precision is one digit after the decimal point.

This tool has been implemented at the Florida International University on a SUN-4 computer running a UNIX-compatible operating system. The programs were written mostly in the C language. The database design descriptions are automatically produced in a publication ready form using the DITROFF text processing package. On-screen graphic output is generated in POSTSCRIPT (particularly, the automatically drawn diagrams of semantic schemas). The current DBMS interface is to the ORACLE system. The following students participated in the tool's implementation and helped in this research: Michael Alexopoulos, Carlos Ibarra, Alok Jain, Ravichandra Kallem, Ranjana Kizakkevariath, Tim Riley, Tatiana Shoshkina, and Eugeni Zabokritski.

5. EXAMPLE

This example describes a database that has been developed for the Hydrology Division of the Everglades National Park. (Actually this application is a self-contained sub-application of a larger database covering various activities of the Park and consisting of more than 1000 categories, relations, and attributes, all of which is managed by our tool.) The following are fragments of the design report generated by the tool.

5.1. Semantic Analysis

5.1.1. Hydrology stations

![Diagram of Hydrology Station]

**HYDROLOGY STATION**

- **station-id**: Char(15) key
- **station-description**: String
- **station-location-north**: 2746840..2865840
- **station-location-east**: 446880..563280
- **location-tolerance**: 0..1000

![Diagram of Fixed Station]

**FIXED STATION**

- **housing-descriptor**: String
- **platform-height**: 0.00..10.00
- **location-north**: 2746840..2865840
- **location-east**: 446880..563280
- **platform-height**: 0.00..10.00

![Diagram of Discontinuous Station]

**DISCONTINUOUS STATION**

the discontinuous station

(m:1,total)

![Diagram of Station Continuity Period]

**STATION CONTINUITY PERIOD**

- **begin-date**: Date key/2
- **end-date**: Date

Figure 5-1. Stations.

HYDROLOGY-STATION — category. (A catalog of hydrology stations which reside within the Park.)
FIXED-STATION — subcategory of HYDROLOGY-STATION. (A hydrology station which is housed in a permanent structure.)

DISCONTINUOUS-STATION — subcategory of FIXED-STATION. (A fixed hydrology station which collects data only for specific intervals of time.)

STATION-CONTINUITY-PERIOD — category. (A catalog of periods during which a discontinuous station is active and various data is collected.)

the-discontinuous-station — relation from STATION-CONTINUITY-PERIOD to DISCONTINUOUS-STATION (m:1,total). (The discontinuous station which was active for periods of time collecting data.)

station-id — attribute of HYDROLOGY-STATION, range: Char(15) (key).

station-description — attribute of HYDROLOGY-STATION, range: String (m:1). (English name or designation of the station.)

station-location-north — attribute of HYDROLOGY-STATION, range: 2746840 .. 2865840 (m:1). (UTM north coordinate of a hydrology station.)

station-location-east — attribute of HYDROLOGY-STATION, range: 446880 .. 563280 (m:1). (UTM east coordinate of a hydrology station.)

location-tolerance — attribute of HYDROLOGY-STATION, range: 0..1000 (m:1). (Tolerance of the location of a station, in feet. A value X assigned to this attribute means that the tolerance is +/-X feet.)

housing-descriptor — attribute of FIXED-STATION, range: String (m:1). (Description of the housing of a fixed station.)

platform-height — attribute of FIXED-STATION, range: 0.00 .. 10.00 (m:1). (The height of the station platform from the water surface, in feet.)

location-north — attribute of FIXED-STATION, range: 2746840 .. 2865840 (m:1). (UTM north coordinate of the benchmark which corresponds to a fixed station.)

location-east — attribute of FIXED-STATION, range: 446880 .. 563280 (m:1). (UTM east coordinate of the benchmark which corresponds to a fixed station.)

platform-height — attribute of FIXED-STATION, range: 0.00 .. 10.00 (m:1). (The difference between the height of the station platform and the height of its corresponding benchmark, in feet.)

begin-date — attribute of STATION-CONTINUITY-PERIOD, range: Date (key/2). (The date during which a discontinuous station was activated and started the generation of data for some parameters.)

end-date — attribute of STATION-CONTINUITY-PERIOD, range: Date (m:1). (The date during which a period of activation for some discontinuous station ended.)

5.1.2. Relational schema of the application

<table>
<thead>
<tr>
<th>HYDROLOGY-STATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>station-id-key:Char(15) 1:1; station-description:String; location-tolerance:0..1000; station-location-east:446880..563280; station-location-north:2746840..2865840; is-discontinuous-station:Boolean; is-fixed-station:Boolean; location-east:446880..563280; location-north:2746840..2865840; housing-descriptor:String; platform-height:0.00..10.00; platform-height:0.00..10.00;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STATION-CONTINUITY-PERIOD</th>
</tr>
</thead>
<tbody>
<tr>
<td>of-station-id-in-key:Char(15); begin-date-in-key:Date; end-date:Date;</td>
</tr>
</tbody>
</table>

The following are some of the integrity constraints automatically generated during schema conversion.

(for every x in HYDROLOGY-STATION: if x.is-discontinuous-station then x.is-fixed-station) and

(for every x in HYDROLOGY-STATION: if not x location-east null then x.is-fixed-station) and

(for every x in HYDROLOGY-STATION: if not x location-north null then x.is-fixed-station) and

(for every x in HYDROLOGY-STATION: if not x housing-descriptor null then x.is-fixed-station) and

(for every x in HYDROLOGY-STATION: if not x platform-height null then x.is-fixed-station) and

(for every x in HYDROLOGY-STATION: if not x platform-height null then x.is-fixed-station) and

(for every x in HOURLY-STAGE: exists y in HYDROLOGY-STATION: x.hourly-produced-by-station-id-in-key = y.station-id-key) and
(for every x in \textit{STATION-CONTINUITY-PERIOD}: exists y in \textit{HYDROLOGY-STATION}: x.of-station-id-in-key = y.station-id-key and y.is-discontinuous-station)

References


© 1994 by Knowledge Systems Institute

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher.

ISBN 0-9641699-0-8 (paper)

Additional copies can be ordered from
Knowledge Systems Institute
3420 Main Street
Skokie, IL 60076
USA

Printed in the United States of America