# A TRACE & UNIFICATION GRAMMAR FOR CHINESE

Hans Ulrich Block

Siemens AG, Corporate Research, ZFE IS INF 23

Otto Hahn- Ring 6, D-8000 München 83

block@ztivax.uucp

Ping Peng

Department of Computer Science

Courant Institute of Mathematical Sciences

New York University

715 Broadway, # 715, New York, NY 10003

peng@cs.nyu.edu

## ABSTRACT

This paper presents a design and an implementation of a unification grammar for Chinese. Furthermore the Chinese grammar is designed as a reversible grammar which serves both parsing and generation. The Chinese grammar is developed under the system of Trace & Unification Grammar that compiles the grammar into an efficient parser and an efficient generator. The implementation shows that a set of Chinese grammar rules used for parsing and generation can be stated elegantly by the unification. Some examples illustrate how to formulate Chinese sentences by reversible grammar rules.

**KEYWORD:** Chinese grammar, Reversible grammar, Unification formalism.

# 1 Introduction

During recent years there has been a growing interest in NL systems that can be used for both parsing and generation. The ideas of a unification grammar that allows for a declarative description of language have made it possible to use the same grammar for both tasks. The main goal of designing a grammar then is to describe a relation between normalized (semantic) representations and language strings. If a grammar can be used in both directions of parsing and generation, we call it a "reversible grammar".

This paper discusses the design of a Chinese reversible grammar and describes its implementation in the system "Linguistic Kernel Processor" developed by Siemens AG, Corporate Research [3]. It has been tested as one component of a machine translation system "Multilingual Conversation Interpreter" which translates dialog-style texts between any pair of languages among English, German, Chinese and Swedish [2].

The reversibility of a grammar requires consideration of two aspects. One is the different procedural interpretations of a grammar in parsing and generation. This can be handled by a mechanism which automatically associates a control interpretation with each of the two opposite directions of computation (see [12], [11], [3] for detailed descriptions). The other one is a way how to formulate a natural language sentence by a grammar so that the grammar can be used both in a process of parsing and in a process of generation. This paper focuses on the second aspect of designing a Chinese grammar.

The mechanism with which the reversible Chinese grammar is written is the "Linguistic Kernel Processor". It provides natural language grammar writers with a tool for designing a reversible grammar, which serves for both natural language sentence parsing and generation. The formalism adopted by the "Linguistic Kernel Processor" is a variant of Unification Grammar combined with "movement rules" based on Government & Binding Theory, called "Trace & Unification Grammar" (TUG). A set of Chinese grammar rules written in this formalism are stated declaratively as context-free productions and PATR-II style feature equations. Context-free productions describe the surface structure of Chinese language strings. A mechanism of "movement rules" is built into production rules to

specify discontinuous dependencies within a language string. Feature equations specify a relation among features of phrases within a production. Unification is prescribed as the sole operation on the feature equations to make a bi-directional computation possible. The equations play roles both in composition of a semantic representation of a phrase from its children during parsing and in decomposition of a semantic representation of a phrase into its children during generation. The mechanisms of feature typing, mixing of attribute-value pairs and Prolog-terms unification, macros, and general disjunctions combine with feature equations to increase flexibility in information-combining and information passing, as well as syntactical and semantical composition and decomposition. The written Chinese reversible grammar is then compiled by the system into a LR parser [13] and a semantic-head-driven generator [10] to enhance the dynamic performance of the parser and the generator (See [3] for details of the introduction.) In the following chapters, we will first give a description of the TUG Formalism, then describe the basic features of the Chinese grammar and finally give some examples of paraphrases generated by the system for Chinese sentence inputs.

# 2 The TUG Formalism

The design of Trace and Unification Grammar has been guided by the following goals:

- **Perspicuity**. We are convinced that the generality, coverage, reliability and development speed of a grammar are a direct function of its perspicuity, just as programming in Pascal is less error-prone than programming in assembler. In the optimal case, the grammar writer should be freed of reflections on how to code things best for processing but should only be guided by linguistic criteria. These goals led for example to the introduction of unrestricted disjunction into the TUG formalism.

- **Compatibility to GB Theory**. It was a major objective of the LKP to base the grammar on well understood and motivated grounds. As TUG was originally applied to German and most of the newer linguistic descriptions on German are in the

framework of GB theory, it was designed to be somehow compatible with this theory though it was not our goal to "hardwire" every GB principle.

- **Efficiency**. As the LKP is supposed to be the basis of systems for interactive usage of natural language, efficiency is a very important goal. Making efficiency a design goal of the formalism led e.g. to the introduction of feature types and the separation of the movement rules into head movement and argument movement.

The basis of TUG is formed by a context free grammar that is augmented by PATR II-style feature equations. Besides this basis, the main features of TUG are feature typing, mixing of attribute-value-pair and (PROLOG-) term unification, flexible macros, unrestricted disjunction and special rule types for argument and head movement.

## 2.1 The framework

As a very simple example we will look at the TUG version of the example grammar in [8].

```
% type definition
s       => f.
np      => f(agr:agrmnt).
vp      => f(agr:agrmnt).
v       => f(agr:agrmnt).

agrmnt => f(number:number,person:person).

number => {singular,plural}.
person => {first,second,third}.

% rules
s ---> np, vp |
     np:agr = vp:agr.
```

```
vp ---> v, np |
     vp:agr = v:agr.

% lexicon
lexicon('Uther',np) |
     agr:number = singular,
     agr:person = third.
lexicon('Arthur',np) |
     agr:number = singular,
     agr:person = third.
lexicon(knights,v) |
     agr:number = singular,
     agr:person = third.
lexicon(knight,v) |
     ( agr:number = singular,
       ( agr:person = first
       ; agr:person = second
       )
     ;
       agr:number = plural
     ).
```

There are two main differences from PATR II in the basic framework. First, TUG is less flexible in that it has a "hard" context free backbone, whereas in PATR II categories of the context free part are placeholders for feature structures, their names being taken as the value of the cat feature in the structure. Second, TUG has a strict typing. For a feature path to be well defined, each of its attributes has to be declared in the type definition.

Besides defined attribute-value-pairs, TUG allows for the mixing of attribute-value-pair unification with arbitrary structures like PROLOG terms using a back-quote notation. This can be regarded as the unificational variant of the BUILDQ operation known from ATNs. As an example consider the following lexicon entry of *each* that constructs a predicate logic notation out of `det:base`, `det:scope` and `det:var`.

```
lexicon(each,det) |
     det:sem =
          'all(det:var,det:base ->
                         det:scope)
```

The usefulness of this feature for the construction of semantic forms will be shown in the section on the Chinese grammar.

TUG provides templates for a clearer organization of the grammar. The agreement in the above mentioned grammar might have been formulated like the following:

```
agree(X,Y) short_for
     X:agr = Y:agr.
...
s ---> np, vp |
     agree(np,vp).
```

TUG allows for arbitrary disjunction of feature equations. Disjunctions and Conjunction may be mixed freely. Besides well known cases as in the entry for *knight* above, we found many cases where disjunctions of path equations are useful, e.g. for the description of the extraposed relative clauses[1].

[1] [4] describes our processing technique for disjunctions.

## 2.2 Features

Features are defined at the beginning of the grammar. Features of a noun phrase (`np`) can e.g. be defined as:

```
np => f(semantics:sem, class:npclass, cmw).
```

By this definition, a noun phrase has three features. The feature `semantics` carries a semantic representation of the noun phrase. The feature `class` indicates a user defined semantic classification to which the noun phrase belongs, which helps to disambiguate syntactic structures of sentences. The feature `cmw` specifies that a designated classifier (see the discussion of 3.2. 1) is required by the noun phrase. Strict typing is used for the definitions of `semantics` and `class`. In operations on features, values of `semantics` and `class` are restricted to be an element of the pre-defined set `sem` and `npclass` correspondingly.

Features are used in grammar rules. The symbol `':'` is used as an infix operator for feature indexing. For example, `np:class` should be read as a value of the feature `class` of the noun phrase `np`.

## 2.3 Movement rules

Besides these more standard UG-features, TUG provides special rule formats for the description of discontinuous dependencies, so called "movement rules". Two main types of movement are distinguished: argument movement and head movement. The format and processing of argument movement rules is greatly inspired by [5] and [6], the processing of head movement is based on GPSG like slash features.

### 2.3.1 Head Movement

A head movement rule defines a relation between two positions in a parse tree, one is the landing site, the other the trace position. Head movement is constrained by the condition

that the trace is the head of a specified sister (the root node) of the landing site[2]. Trace and Antecedent are identical with the exception that the landing site contains overt material, the trace doesn't. Suppose, that v is the head of vk, vk the head of vp and vp the head of s, then only the first of the following structures is a correct head movement, the second is excluded because np is not head of vp, the third because antecedent and trace are unequal.

$[_{s'}$ $v_i$ $[_s$ ... $[_{vp}$ ...

$[_{vk}$ ... trace(v)$_i$ ...]...]...]...]

$[_{s'}$ $np_i$ $[_s$ ... $[_{vp}$ trace(np)$_i$ ...

$[_{vk}$ ... v ...]...]...]

$[_{s'}$ $np_i$ $[_s$ ... $[_{vp}$ ...

$[_{vk}$ ... trace(v)$_i$ ...]...]...]...]

To formulate head movement in TUG the following format is used. First, a head definition defines which category is the head of which other.

```
v is_head_of vk.
vk is_head_of vp.
vp is_head_of s.
```

Second, the landing site is defined by a rule like

```
s' ---> v+s | ...
```

To include recursive rules in the head path, heads are defined by the following head definitions. In a structure $[_M$ $D_1$ ... $D_n]$ $D_i$ is the head of M if either $D_i$ **is_head_of** M is defined or $D_i$ has the same category as M and either $D_i$ **is_head_of** X or X **is_head_of** $D_i$ is defined for any category X.

Head movement rules are very well suited for a concise description of the positions of the finite verb in German (sentence initial, second and final) as in

[2]Here, "head of" is a transitive relation s.t. if x is head of y and y is head of z then x is head of z.

*Hat$_i$ der Mann der Frau das Buch gegeben t$_i$?*

Has$_i$ the man the woman the book given t$_i$

*Der Mann hat$_i$ der Frau das Buch gegeben t$_i$*

The man has$_i$ the woman the book given t$_i$

*... daß der Mann der Frau das Buch gegeben hat*

... that the man the woman the book given has

All that is needed are the head definitions and the rule that introduces the landing site[3].

### 2.3.2 Argument Movement

Argument movement rules describe a relation between a landing site and a trace. The trace is always c-commanded by the landing site, its antecedent. Two different traces are distinguished, anaphoric traces and variable traces. Anaphoric traces must find their antecedent within the same bounding node, variable trace binding is constrained by subjacency, e.a. the binding of the trace to its antecedent must not cross two bounding nodes. Anaphoric traces are found for example in English passive constructions [$_s$ [$_{np}$ The book of this author]$_i$ was read t$_i$] whereas variable traces are usually found in wh-constructions and topicalization. Similar to the proposal in [5], argument movement is coded in TUG by a rule that describes the landing site, as for example in

```
s2 ---> np:ante<trace(var,np:trace), s1 |
     ante:fx = trace:fx,
     ...
```

[3]On a first glance, one might be tempted to consider head movement as a speciality of German syntax. This is not necessarily true, as it can e.g. also be used for the description of English Subj-Aux inversion.

*Peter has been reading a book*

*Has$_i$ Peter t$_i$ been reading a book*

As to Chinese syntax, the existence of head movement remains unclear at the moment.

This rule states that np:ante[4] is the antecedent of an np-trace that is dominated by s1. This rule describes a leftward movement. Following Chen's proposal, TUG also provides for rightward movement rules. A rightward movement rule might look like this.

```
s2 ---> s1, trace(var,np:trace)>np:ante |
     ante:fx = trace:fx,
     ...
```

The first argument in the trace-term indicates whether the landing site is for a variable (var) or for an anaphoric (ana) trace. Other than head movement, where trace and antecedent are by definition identical, the feature sharing of argument traces with their antecedents has to be defined in the grammar by feature equations (ante:fx = trace:fx, ...). Furthermore, it is not necessary that the antecedent and the trace have the same syntactic category. This is important for e.g. the rule for pronoun fronting in German might which can be stated along with rules like the following:

```
spr ---> pron<trace(ana,np), s | ...
```

The current version of the formalisms requires that the grammar contains a declaration on which categories are possible traces. In such a declaration it is possible to assign features to a trace, for example marking it as empty:

```
trace(np) | np:empty = yes.
```

Bounding nodes have to be declared as such in the grammar by statements of the form

```
bounding_node(np).
bounding_node(s) | s:tense = yes.
```

[4]The notation Cat:Index is used to distinguish two or more occurrences of the same category in the same rule in the equation part. :ante and :trace are arbitrary names used as index to refer to the two different nps.

As in the second case, bounding nodes may be defined in terms of category symbols and features[5]. Typical long distance movement phenomena are described within this formalism as in GB by trace hopping. Below is a grammar fragment to describe the sentence *Which books$_i$ do you think $t_i$ John knows $t_i$ Mary didn't understand $t_i$*:

```
bounding_node(s).
bounding_node(np).

s1 ---> np<trace(var,np), s | ...
s ---> np, vp | ...
s ---> aux, np, vp | ...
np ---> propernoun | ...
np ---> det, n |
vp ---> v, s1 | ...
vp ---> v, np | ...

trace(np).
```

The main difference of argument movement to other approaches for the description of discontinuities like extraposition grammars [7] is that argument movement is not restricted to nested rule application. This makes the approach especially attractive for a scrambling analysis of the relative free word order in the German *Mittelfeld* as in

*Ihm$_i$ hat$_j$ das Buch$_k$ keiner $t_i$ $t_k$ gegeben $t_j$*. The usefulness of this feature for the description of Chinese is described in [5] and [6].

# 3 Description of Chinese

In designing a reversible grammar, it is important to find an adequate description of linguistic knowledge that we would like to use for both parsing and generation. To accom-

[5]Currently, only conjunction of equations is allowed in the definition of bounding nodes.

plish this, it is necessary to have a grammar formalism be completely declarative and its interpretation order-independent. On the other hand, grammar rules should be tailored accurately, not only to provide large coverage for a sentence analysis but also to restrict overgeneration of language strings.

Concerning the Chinese language, its sentences are less structured than those of western languages. There are no relative pronouns or inflections. Sometimes, an active sentence and a passive sentence may share the same surface structure. Compare the following two sentences:

1. The English sentence "I walked." has a Chinese sentence equivalence:

   wo zou le.
   "I walk"

2. The English sentence "The book is bought." has a Chinese sentence equivalence:

   shu mai le.
   "book buy"

In these two sentences, correct constructions of the syntactic trees and the semantic representations are mainly derived from the lexical semantics within the sentences. Word order in a sentence can be very flexible, though the average length of a sentence is shorter than that in western languages. An object without any inflected marker in a sentence is dislocated frequently. To disambiguate syntactic structures and to build up correct semantic representations, semantic information in lexicon and word orders in sentences play very important roles.

In the Chinese grammar, features and feature structures are defined for each phrase. They carry necessary syntactic, semantic and pragmatic information for parsing and generation. These features are instantiated or passed through feature equations. Furthermore those feature structure are composed or decomposed level by level by feature equations.

## 3.1 A semantic representation

In our Chinese language processing, Quasi Logical Form, which is a contextually-sensitive logical form language [1], is chosen for the semantic representation of a sentence. Several sentences with different surface structures may be mapped into the same semantic representation. For example, Chinese yes/no questions appear regularly in different sentential forms:

- ni mai shu ma?
  "you buy book"
- ni mai bu mai shu?
  "you buy not buy book"
- ni mai shu bu mai?
  "you buy book not buy"
- ni mai shu bu mai shu?
  "you buy book not buy book"

In the analysis, the same semantic representation is produced from any of the above four sentences. In the generation, all of these four sentences are produced from the semantic representation. A tendency in the analysis of sentences is to discard some information which is not related to syntactic and semantic representations. In the above case, information about certain sentential style in a Chinese question is ignored while the information is necessary to designate which one of the above should be generated.

## 3.2 Selected examples

### 3.2.1 Classifiers

In Chinese, when a quantity word is used to describe a quantity of a noun, a classifier which is also called a count measure word must be inserted in between the quantity word and the noun. For an English phrase "one book", its Chinese equivalency is "yi (one) *ben* shu

(book)". Here, *ben* is a classifier associated with a quantity word for describing a quantity of the noun "shu" (book). For another English phrase "one car", its Chinese equivalency is "yi (one) *liang* che (car)". Here, *liang* is a classifier associated with a quantity word for describing a quantity of the noun "che (car)". Classifiers vary with nouns. Each classifier has to match the noun which a quantity word modifies. A selection of a classifier is not determined by s surface structure of a phrase, but by a lexical item of nouns.

The following fragment of grammar rules is used for parsing and generating a noun phrase with a quantity modifier.

```
np ---> cmwp, noun |                                          (1)
        cmwp:form = noun:cmw,
        np:sem = cmwp:sem,
        cmwp:restr = noun:sem.
cmwp ---> quantity, cmw |                                     (2)
          cmwp:form = cmw:form,
          cmwp:sem = 'qterm(quantity:sem,cmwp:restr).

lexicon(yi, quantity)| quantity:sem = 1.
lexicon(ben, cmw) | cmw:form = ben.
lexicon(shu, noun) | noun:sem = shu,
                     noun:cmw = ben.
```

Two points can be observed from the fragment of grammar rules.

1. feature passing and equality testing:

   In order to enforce a semantic restriction upon a classifier and a noun, we use the equation "`cmwp:form = noun:cmw`", which checks whether a value of the feature `form` of `cmwp` is the same as a value of feature `cmw` of `noun` in *rule (1)*. The value of the feature `form` of `cmwp` represents the value of the feature `form` of `cmw`. It is defined in the lexical item "`lexicon(ben,cmw) | cmw:form = ben`" and is passed to `cmwp` through the equation "`cmwp:form = cmw:form`'' in *rule (2)*.

2. classifier generation:

   Since a classifier is not coded into the semantic representation, a classifier generation can not be done with the input semantic representation. The solution in the fragment of grammar rules is to use lexical information. A value of the feature `cmw` is found in a lexical item of noun after the noun is selected. It is then passed to `cmw` through the two equations "`cmwp:form = noun:cmw`" and "`cmwp:form = cmw:form`". Some values which are discarded in a process of parsing but are useful to select a lexical item in a process of generation can be recovered correctly.

### 3.2.2 Topicalization

An usual word order of a Chinese declarative sentence is similar to that in English, that is, subject-verb-objects. Quite frequently, an object can be topicalized. The topicalized object is preceded by a syntactic marker "ba" ("ba" is called a "virtual particle" in Chinese) and is placed between a subject and a verb. The English sentence "I have bought a book." can be interpreted as:

- wo mai shu le.

  (*usual*): subject_verb_object

- wo ba shu mai le.

  (*topicalized*): subject_ba_object_verb

Overgeneration may arise. The problem is that an object topicalization sentence is allowed when the verb in the sentence has two objects or an adjunct such as the particle "le" (completed). For instance, a topicalized sentence "wo ba shu mai." is not adequate in daily dialog except in a Peking opera. To overcome the overgeneration, a feature is set up to detect an appropriate form. The following fragment of grammar rules shows how the feature play the role.

```
s ---> np, vp.                    (1)
vp ---> db.                       (2)
```

```
vp ---> ba, np<trace(ana,np:np_trace),db |          (3)
        db:weight = heavy.
db ---> v, np |                                     (4)
        db:weight = v:weight.
db ---> v, np, np |                                 (5)
        db:weight = heavy.
v ---> verb |                                       (6)
       v:weight = light.
v ---> verb, le |                                   (7)
       v:weight = heavy.
```

Here, a left movement rule gives the landing site of the np in rule (3). The similar treatment of the movement transformation has been proposed in [6]. An interesting point in this grammar rule is that the same movement rule used for parsing a ba-structure sentence is used for generating a surface structure of ba-sentence. In order to overcome the overgeneration mentioned above, the feature "*weight*" is created in a verb phrase. A value of the feature "*weight*" indicates the adequate surface structure that should be generated from a quasi-logic form. When the value of the feature "*weight*" is "*light*", the possibility of generating a ba-structure is eliminated. Only when the value of the feature "*weight*" is "*heavy*", a ba-structure can be derived from an internal semantic representation, that is, a quasi-logic form in our grammar.

# 4 Using a Paraphraser for grammar testing

In the above system, the declarative content of the Chinese grammar is shared by both the parser and the generator. The Chinese grammar is compiled dually into a parser and a generator automatically. The parser transforms a Chinese sentence into a quasi-logic form which we use for our internal semantic representations of languages in our machine translation system. The generator produces a Chinese sentence from

a quasi-logic form. We define the predicate *paraphrase(X, Y)* to show the reversible computation. The first argument X of the predicate *paraphrase* is bound to an input of a Chinese sentence. The second argument Y of the predicate *paraphrase* is any output of a Chinese sentence generated from the quasi-logic form to which the input Chinese sentence is transformed.

```
?- paraphrase(['wo^','ke^yi','bangzhu','ni^','ma'],O).
S E M A N T I K:
ynq(ke3yi(bangzhu(qterm(qcat(-,-,ex,sg),X,[event,X]),
      a_term(ref(_,th,perspron,1,_,sg,_),Y,[personal,Y]),
      a_term(ref(_,rh,perspron,2,_,sg,_),Z,[personal,Z]))))
O = ['wo^','ke^yi','bangzhu','ni^','ma'];
O = ['wo^','ke^yi','bu`','ke^yi','bangzhu','ni^'];
O = ['wo^','ke^yi','bangzhu','ni^','bu`','ke^yi'];
O = ['wo^','ke^yi','bangzhu','ni^','bu`','ke^yi','bangzhu','ni^'];

no
```

This example shows how the sentence

- wo keyi bangzhu ni ma?
  "Can I help you?"

is analysed and is generated from its semantic representation. The generator enumerates all possible paraphrases that are covered by the grammar for one semantic structure.

# 5 Conclusion

We have discussed some issues in designing a reversible grammar. We have shown how a reversible Chinese grammar can be designed under the formalism of Trace

& Unification Grammar. The examples illustrate how some Chinese language phenomena can be handled by the Chinese grammar. There are about one hundred grammar rules in our current Chinese grammar. It takes 0.2 to 1.2 seconds to parse and generate a sentence up to 10 words. The result shows that a reversible Chinese grammar not only is possible but also performs effectively in practical applications.

**References.**


[1]: Alshawi, H. "Resolving Quasi Logical Forms". *Computational Linguistics*, Vol. 16, pp. 133-144, 1990.

[2]: Alshawi, H., H. U. Block, D. Carter, B. Gambäck, R. Hunze, P. Peng, M. Rayner, S. Schachtl and L. Schmid. "Communication Multilingue par Forme Quasi Logique". *Proc. Expert Systems and their Applications, Avignon*, 1991.

[3]: Block, H. U. "Compiling Trace & Unification Grammar for Parsing and Generation". *Proc. The Reversible Grammar Workshop, ACL, 1991.*

[4]: Block, H. U. and L. A. Schmid. "Using Disjunctive Constraints in a Bottom-Up Parser". *Forthcoming.*

[5]: Chen, H.-H., I-P. Lin and C.-P. Wu. "A new design of Prolog-based bottom-up Parsing System with Government-Binding Theory". *Proc. 12th International Conference on Computational Linguistics* (COLING-88), pp. 112-116, 1988.

[6]: Chen, H.-H. "A Logic-Based Government-Binding Parser for Mandarin Chinese". *Proc. 13th International Conference on Computational Linguistics* (COLING-90), pp. 1-6, 1990.

[7]: Pereira, F. "Extraposition Grammar". *Computational Linguistics* Vol. 7, pp. 243-256, 1981.

[8]: Shieber, S.M. "The design of a Computer Language for Linguistic Information". *Proc. 10th International Conference on Computational Linguistics* (COLING-84), pp. 362-366, 1984.

[9]: Shieber, S.M. "A Uniform Architecture for Parsing and Generation". *Proc. 12th International Conference on Computational Linguistics* (COLING-88), pp. 614-619, 1988.

[10]: Shieber, S.M., G. van Noord, F.C.N. Pereira and R.C. Moore . "Semantic-Head-Driven Generation". *Computational Linguistics*, Vol. 16, pp. 30-43, 1990.

[11]: Strzalkowski, T. and Ping Peng. "Automated Inversion of Logic Grammars for Generation". *Proc. Conf. of the 28th Annual Meeting of the ACL*, (ACL-90) pp. 212-219, 1990.

[12]: Strzalkowski, T. "How to Invert a Natural Language Parser into an Efficient Generator: An Algorithm for Logic Grammars". *Proc. 13th International Conference on Computational Linguistics* (COLING-90), pp. 347-352, 1990.

[13]: Tomita, M. *Efficient Parsing for Natural Language: A fast Algorithm for Practical Systems.* Boston: Kluwer Academic Publishers, 1986.