NATURAL LANGUAGE, SEMANTIC ANALYSIS AND INTERACTIVE FICTION

Graham Nelson

St Anne's College, Oxford

[10 April 2005; revised 10 April 2006⋆]

This is an account of theoretical issues which came out, almost unbidden, from a practical test of the following hypothesis: that *the natural language in which to write interactive fiction is natural language*. IF is a form of creative writing impossible before the development of computing, but whose 30-year history has seen a flourishing of experimentation if not mainstream acceptance (except in an early commercial phase): the author creates an imaginary textual world which can actively be explored by a "reader", or "player", directing the actions of a protagonist. Such works have hitherto been created as if computer programs, using specially adapted programming languages (see for instance Nelson (2001)), but the Inform 7 project aims to replace such syntax with natural language: specifically, a subset of English. This change proved far more radical than had initially been expected, and it became clear that semantic analysis and related branches of linguistics were of great relevance to practical issues of how design systems for IF should work.

The Inform 7 project began in 2002 as an experimental higher-level layer on top of the existing Inform system for designing IF, now in use since 1993. At time of writing, an application for Mac OS X and Windows is just about to be published as a public beta.

This paper is divided into two. Part 1, 'Naturality in Practice', describes and explores the motivation for the three conceptually new aspects of Inform 7: the user interface (§1a), the shift to natural language (§1b) and the adoption of rule-oriented rather than object-oriented design (§1c). Part 2, 'Naturality in Theory', draws on the readings in semantics which guided the Inform project, discussing in turn conceptual semantics (§2a), predicate logic (§2b) and model theory (§2c). The general slogan here is that the writing of IF is a form of narration; that a system for writing IF can be judged by the range of meaning it narrates; and that semantic analysis, the branch of linguistics concerned both with narrow and broad questions of meaning, is therefore of central importance to theories of IF.

In Part 1, I argue that the three major shifts described are all moves toward a more natural

⋆ This paper was written as a contribution to the forthcoming *IF Theory* book, and though an interim report it is also a manifesto which remains a fair statement of the project's ideology. The footnotes were added later in 2005, and the material then reorganised and redrafted in April 2006.

kind of writing. "Writing" is an ambiguous term: it might equally well mean a set of markings on paper, the activity of putting words together, or the prose which results: and for the same reason we must be precise in what we mean by "programming IF", and in what we are claiming about it. First I suggest that the activity of programming IF is a form of dialogue between programmer and computer to reach a state with which both are content, and that it is not unlike the activity of playing IF, also a continuing dialogue in which the computer rejects much of what the user tries. Secondly, the place where this activity goes on is not conceptually a single page of typing paper, as would be offered by a word-processor, but is more like a book of translations presented in parallel text: with facing pages, one written by the programmer and one by the computer. Thirdly, the program which results from all this activity (the "source text") is a description of an imaginary situation which extends through time – a story, in fact. The central idea of Part 1 is that a "natural" system for IF is one in which all three of these comparisons are tautologies: that the activity is explicitly a dialogue, that the user interface looks and behaves like a book with facing pages, and that the source text reads like a narrative.

In Part 2, I argue that the formal study of what is conceptually natural – that is to say, of semantics in the broadest sense used in linguistics – is a useful perspective on questions of how IF design systems should work. Natural languages make story-tellers of us all, and are well-adapted to the description of situation and event. Semantic analysis may be able to tell us what concepts and structures within natural language give it such facility in story-telling: looking for the presence or absence of these features in programs for writing IF may provide an insight into why certain kinds of IF are written but not others. Comparison with the literature of semantics may also help to question unconscious assumptions built in to systems for IF: for instance, are containers as important as we seem to think? Do we really perceive the world in terms of objects which inherit properties from classes, or is that a conceit of computer programming? What should be part of the core functionality of a system for IF, and what can be relegated to third-party extensions, or left for writers to sort out for themselves? How shall we judge such questions of what matters most?

I wish to acknowledge, and those four words are woefully inadequate, the help I have received with the Inform 7 project from people who have at various stages contributed to its ideas fully as much as their practical expression: and especially Emily Short, Andrew Plotkin, Sonja Kesserich, Andrew Hunter and David Kinder. Tendentious opinions here are my own, but I could not have formed any opinion without the last three years of discussion and collaborative effort, and I particularly wish to thank all those who have read and commented on drafts of this paper.

Part 1. Naturality in Practice

§1a. A humanising interface

Early builds of Inform 7 coincided with the 20th anniversary of the Apple Macintosh user interface (1984). I had begun the project by collecting together notes into a self-styled *Book of Inform*, my version of the *Book of Macintosh* collated around 1982 by Jef Raskin (1943–2005): a mixture of the practical and impractical, and a description rather than a blueprint, and which was free to look nothing like the final product. The Macintosh team drew inspiration from the iconography and shape of road-signs, the function of the bicycle, the office environment and the industrial design of cars: the aim was to make a computer a domestic appliance as natural as, say, a kettle (see for instance the recent memoir Hertzfeld (2004)). Similarly, the Book of Inform aimed to describe a radically humanising interface for the writing of interactive fiction (IF). My earlier program, Inform 6, had been a computer programmer's tool which aimed to be welcoming to creative writers: this aspired to be the other way around, and its guiding metaphor would be that of the interactive book. In 2003 I had the great good fortune to recruit Andrew Hunter, author of the best-interfaced IF interpreter for Mac OS X ("Zoom"), to the project: the reference implementation of the interface is entirely his work. David Kinder then took on the coding of the corresponding Windows interface, which was no small feat since essentially none of Andrew's code could be used there, and the entire system had to be written afresh.

To deal first with what was being abolished, the *Book of Inform* tried to remove the computer's filing system from the picture. Setting up a new Inform 6 project, and installing Inform 6, is a nuisance: it means creating a directory, working out commands to compile source into a story file, then to play it, run scripts through it, and so forth. This is discouraging the first time, tedious subsequently. The shortest legal Inform 6 source – the equivalent of that prototypical program, "hello world" – involves three references to filenames and is complicated enough that the books on Inform 6 suggest that newcomers copy it out blindly. By contrast, Inform 7 projects are automatically managed and look like single objects on the host computer. The shortest legal source reads: "Home is a room." Reference to other people's code – any modern system for IF must recognise the highly collaborative nature of IF design today – is made by the name of what is being included, and whom it is by. Thus the source might read, early on:

```
Include the Automatic Door Rules by Emily Short.
```

rather as a book might be prefaced by a list of acknowledgements (and indeed Inform uses it to place just such a list in the compiled game). No filenames appear, nor any platform-specific references.

A project is a single book, not a docket of intermediate states in disparate formats and with cryptic names. But if it were one long endless stream of prose, it would quickly become disorganised (as early testing made abundantly clear). Most computer programs of any size are internally organised by being divided up into separate source files by function, but this seemed wrong for Inform because it took us back to filing systems. A partial solution came from "literate programming", Donald Knuth's scheme for interleaving code and commentary (and indeed parts of the Inform program itself use Knuth's CWEB

system: Knuth and Levy (1994)). Though Knuth's writings on programming stylistics, conveniently gathered in Knuth (1992), contain little systematic thought and are essentially rooted in the debates of what is now a bygone age (structured programming: grail or poisoned chalice?), they are nevertheless well worth reading.[1] His essential remedy was to reconcile program with book by promoting a form of program easy to typeset, so that it would always have a dual existence: a human-readable one, and a computer-executable one, both continuously kept up to date. Inform goes along with this in dividing code into paragraphs, and also (as we shall see) in indexing, but ultimately adopts the same solution that books have used since the *Iliad*: it divides the source text up into sections, chapters, parts, books and volumes, allowing for a hierarchy of headings and subheadings as elaborate or simple as the author prefers. There is no compulsion to use headings, but a number of incentives are offered to persuade authors into the habit: automatic contents listings, better-signposted Problem messages, and so on.

Some approaches to "interactive fiction for the non-programmer" have imitated database packages in displaying and editing projects as wallcharts, in which the various functionalities are boxes connected by lines rather as photographs of suspects are joined by threads on police noticeboards. This is seductive for object-based IF, because those boxes and lines can be related to the coarse structure of the work (map connections, most obviously). But it fragments the writing into tiny pieces. Some creative writers thrive on this – one thinks of Elizabeth Bishop hanging half-stanzas out on the washing line slung across her study – but few of us would choose to draft a novel on ten thousand Post-It notes. Fewer still would wish to edit or revise a novel written in this way, and such approaches to IF make second thoughts and bug reports tiresome to act upon.

Inform instead presents the user with an interface intended to look like an open book with facing pages. The author's work appears in full on the left-hand page, while its consequences appear on the right. This feels natural to someone reading left to right, and agrees with the conventional layout of cartoon panels. The page spread suits today's increasing use of LCD monitors in the aspect ratio 16:10, but several of Inform's testers used 4:3 monitors equally well: the gutter between the two pages slides freely left or right, closing the one up and expanding the other, so that the user can decide which should occupy the greater space. Both pages contain text which is word-wrapped in real time as the pages are resized.

Both pages contain text, rendered in a variable-pitch font with strong anti-aliasing: a font chosen for the legibility of running prose, rather than a typical programmer's text editor font, which uses a fixed pitch to preserve vertical alignments and over-stresses punctuation marks. Although syntax-colouring is offered, the result is less kaleidoscopic than in most programming environments since Inform source text has few lexical categories: there is only "quoted text", unquoted text and comment [in square brackets].

The facing pages are the forum for interplay between the writer and the computer. Inevitably this dialogue is led by the human, typing the source text on the left, and the computer's part is reactive, producing replies. In most languages programming has a code-compile-test cycle, where the compiler often rejects the code and forces the author to make corrections. This is not unlike the experience of playing through IF: think of something, try it out, make progress. Most IF critics agree that an enjoyable game requires

1. The day after writing this somewhat slighting remark about Professor Knuth, I was introduced to him, and he really couldn't have been a nicer guy. It has to be said that CWEB today is a mess, just the same.

a lively, keep-things-moving response to incorrect guesses, because guesses are more often wrong than right. But compiler programmers persist in regarding incorrect input as an aberrant circumstance in which it is inappropriate to make any judgement of the quality of the output. Compilers such as gcc also more often reject input than accept it, but do so with error messages that are nasty, brutish and short. Error messages aim at precision in characterising the exact symptom of failure, but do so in terms of the compiler's own internal data structures or methods. For instance, gcc's error:

> main.c:81: request for member 'count' in something not a structure or union

is factually correct but implies, untruthfully, that the problem lies with the structure; in fact the structure is fine, but the wrong request to access it has been used ('.' not '->'). Similarly,

> main.c:175: parse error at end of input

commonly occurs when a close brace has been missed out, and while it is true that this was detected at the end of input, the problem is almost certainly somewhere else entirely. Such errors describe an accident in terms of the evidence, not the proximate cause. It is arguable that only final object code matters, which may be used by millions of people, and not the passing inconvenience of the programmer, so that maintenance on gcc should concentrate on code optimisation rather than tidying up error messages: but consider what improvements to that final code might be made by a programmer whose time is not being wasted.[2]

Indifference to the convenience of error messages is well exemplified by the current GNU standards document (Stallman, 2004), which covers the punctuation of error messages in some detail while, significantly, offering no guidance on their stylistics, other than to offer the startling recommendation that:

> In error checks that detect 'impossible' conditions, just abort. There is usually no point in printing any message... Whoever wants to fix the bugs will have to read the source code and run a debugger.

Inform does not follow this advice: it accepts the likelihood that its own bugs will be encountered by real users and tries to deal with such contingencies as helpfully as possible, at least identifying where the problem has arisen, so that the user can try work-arounds. And in general Inform's errors do not follow the traditional Unix-command-line pattern.[3] They are called Problems, not errors, they are not confined to one line, they are not reported by the line number on which they occur – instead, Inform talks about sections or chapters and makes generous use of quotation – and they include explanatory text which typically gives examples of correct and incorrect usage. The same basic error can result in different Problem messages according to Inform's guess at the most likely way it arose.

2. Matters grow worse with C++, where a minor industry now exists in selling software whose sole task is to interpret gcc's error messages. At Apple's 2005 Developer Conference, an entire session was given over to a lecture on what the dozen newly-introduced gcc 4.0 error messages actually mean. The session was recently posted online, and it is striking how often the speaker says "Well, this almost always means that..."

3. Though the pioneers of Unix would not necessarily have agreed with the culture of ultra-concise errors sometimes attributed to them. Kernighan and Plauger (1976): "...a prompt is given reminding the user how to use the program properly. Better to tell people concisely how to do things right than tell them only that they did something wrong."

Many of these Problem messages have been added in beta-testing, to give more rewarding responses to reasonable but incorrect things tried by the testers: a process which we found strikingly like that of finishing a work of IF, at the stage when the designer adds numerous responses to cover all the unexpected cases which turned up in testing.

The Inform "coding cycle" consists of typing or amending the source in the left-hand page, then clicking the Go button. Either the source is rejected, in which case the right-hand page responds with a report of the problems found, or it is accepted, in which case the right-hand page begins playing the resulting work of IF. If the Replay button had been clicked instead, the work would automatically play through to its last position, using the same commands. If the author finds a trivial mistake – a spelling error, say – then this can be fixed in the source and the correction verified in a matter of moments.

Leaving aside unpredictable variations based on random-number generation, interactive fiction bears comparison with turn-based games such as chess. This observation motivated an approach to testing IF modelled on the analysis of chess openings, which run together for a while but then diverge. Every Queen's Gambit begins with the same first three moves (1. d4, d5; 2. c4), but then there is a choice, as the next move decides whether we have a Queen's Gambit Accepted (dxc4) or Declined (e6). It would be impractical to study every possible sequence of play, so chess manuals instead contain tables of standard openings. Such "opening books" are essentially built by watching every grandmaster-level game reported in the chess literature and seeing where they innovate. Inform uses the same method: it watches every command sequence typed in during testing to see if this duplicates a previous test, or breaks away. The resulting structure is called the skein, because it is a braiding (or rather a gradual unbraiding) of the possible sequences of commands, which we think of as threads. The skein display allows the author not only to look through all of the games ever played through the fiction in question, but also to replay any of those games, and see what has altered. (The Replay button does this for the most recent thread in the skein.) This aims to make debugging complex works of IF a more reliable business. In large works of IF, small changes in one place often have unforeseen consequences elsewhere, and a major cause of error is the accidental inclusion of one bug while fixing another. The skein quickly grows large, but can be pruned back if the writer chooses, or can be annotated with notes such as "Test falling off cliff" which can be searched for – just as chess opening books have annotations such as "Queen's Gambit".

The final component of Inform is the index which it automatically generates for every project, after each build. At its simplest this is a navigation tool for jumping to the relevant points in the source text – an important consideration in a text the size of a novel. But it is also an aide-memoire with cross-references to the documentation. It offers a choice of viewing the index in a variety of conceptually different ways: by headings and subheadings, like a contents page; by rooms and their contents, in a map intended to follow the style of 1980s printed solution-books for IF; by scenes and their possible sequences, in a corresponding map of chronology; as a taxonomy of the kinds of thing found in the model world for the project (an idea drawn from the SmallTalk class browser of the late 1970s); and as a collation into logical order of all of the various rules. All these viewpoints are valid and Inform makes no judgement about which is pre-eminent. For example, the index of kinds shows, among other things, what objects exist for each kind. This is not the best way to see what the game contains – the map is much clearer – but is just right for an author contemplating (say) a new rule about containers, but worried that it might not

always be appropriate: since the Kinds index lists all the containers existing in the world, one can see at a glance everything to which the new rule would apply.

It may be worth mentioning what was overlooked by the *Book of Inform*, or more specifically two issues which did not really occur to me as non-routine until much later on. First, documentation. The so-called DM4 (Nelson 2001), fourth edition of the Inform 6 manual, aspires to be something of a cult book to its modest readership, a textual "maze of twisty little passages" festooned with arcana and folklore. In so far as it had models, they were Larry Wall's "Camel book" of Perl and Donald Knuth's TeXBook: there is something a little appealing about playing the eccentric inventor, but I perhaps overlooked that neither book – though brilliant, indispensable, and such – is actually much good at its stated purpose. The Inform 7 documentation takes an opposite tack: it tries to be concise, to divide into short screen-readable subsections – it is built into the user interface itself – and not to try the reader's patience over-much with facetiae, nor to try to combine the manual with a history and critical study of IF. The much-criticised "exercises" of the DM4 – actually showcases for surprising possibilities, not pedagogical tests – were replaced by some 260 "examples" which build up into what is described as a "recipe book": these are intended to be imitated and borrowed from. Each contains a complete source text, not an excerpt, and comes with a rapid, automated means of seeing it work. Is the current version of the documentation useful? Is the current version enough fun, come to that?[4] Time will tell: for now, no physical volume is being published in book form.

Another initially unplanned area of Inform 7 was the packaging and publishing of new works of IF: does the design system's responsibility end when the program is compiled? Inform now goes further: it can generate supporting material, and it helps with the bundling up of this material for the eventual player, and in particular with bibliographic data (where it is slightly coercive in an effort to make authors produce works which are easier to archive and browse in databases). As with language design (see §1b below), the analogy with actual books was actively pursued.

With some user interfaces, less is more. Adobe's *InDesign* is a fine program but its plethora of buttons, cursors, palettes, inspectors and toolbars means that, for beginners at least, it might as well be called *InDecision*. Arguably graphic design requires, or at least suits, such an interface. But does IF design? Throughout the experimental period, I would always want fewer buttons and the focus kept to one window, whereas Andrew was more open-minded. Often I would express initial scepticism, but then give in a week later. The feature of which we are least sure is the Inspector, Inform's only subsidiary window: it gathers a collection of functions which benefit from operating without the need to disturb the contents of the main window. The Inspector began simply as an overflow for things which would not fit elsewhere, and for some months I kept it permanently closed (Inform remembers one's preferences in this sort of thing), but I now find it too useful to banish.[5]

4. A serious issue: consider for instance *Why's Poignant Guide to Ruby*, reprinted in e.g. Spolsky (2005): strip cartoons of foxes in chapter 3 of the *Guide* act as a sort of subversive chorus to the ostensibly just-the-facts text, and they keep one reading even when one intended to learn Python instead. At one time, I did want to include classic cartoons in Inform's documentation, too: Gary Larsen's Far Side on "The Curse of Mad Scientist's Block" and Bill Watterson's strip of six-year-old Calvin creating the Universe as one of the "Old Gods" who demands "Sacrifice" – but licensing fees of $400 and endlessly unclear copyrights deterred me.

5. Famous last words. Over the summer of 2005 we moved the Inspector's main selling-point, the search tool, onto the main window, and it is now out of favour again.

Every user interface project needs one curmudgeon who, by doing none of the actual work, forms opinions about features without reference to how much, or how little, effort went into them. It was a great luxury that better programmers than myself allowed me to play this role.

§1b. The adoption of natural language

Natural language as a literal paraphrase of procedural code can suffer from the faults of both, and many programming languages which superficially ape natural language (such as COBOL, or AppleScript) are not convincingly "different" in feel from orthodox coding languages such as C. This may be because they do not contain genuinely linguistic features, but even so, I feel that natural language is easily overlooked as a syntax option in the design of new systems. This is the more odd since many such designs often begin with sketches in "pseudocode": English sentences which approximate what they will one day be coding in more formal ways. The curse of pseudocode is that it is self-consciously *pseudo*-code: we forget that it could also serve as actual prototype syntax.

As an example, consider the Sloan Digital Sky Survey (SDSS), an immense reference database of astronomical observations, and one of the world's largest non-commercial exercises in data warehousing: an imaginative project with an excellent website. In Szalay et al. (2000), a summary of what were then proposed aims, we find an outline of the benefits from new search possibilities:

> Other types of queries will be non-local, like "find all the quasars brighter than r=22, which have a faint blue galaxy within 5 arcsec on the sky". Yet another type of a query is a search for gravitational lenses: "find objects within 10 arcsec of each other which have identical colors, but may have a different brightness".

Here natural language acts as a pseudocode for database-searching programs. In 2006, with SDSS up and running, the public outreach website does allow queries like those envisaged by Szalay et al., but knowledge of SQL programming is required. The following pattern is offered as an example in SDSS Data Release 4 (2006):

```
select
p.objID,p.ra,p.dec,s.z as redshift,w.plate,s.fiberID
from
SpecObj s, PhotoObj p, plateX w
where
p.ObjID=s.bestObjID and w.plateID=s.plateID and
s.z > 4 and s.zConf > 0.95 and s.specClass = 3
```

In pseudocode, this would be "get the Object IDs, positions, redshifts, and plate and fiber numbers of quasars with redshift greater than 4". Which is easier to write without trivial errors causing the search to fail?[6] In a textbook on the use of SDSS, which is more likely to be printed without typographical mistakes or bugs? Which could be quoted as a footnote in a scientific paper, enabling the reader to duplicate the data-set used by the author? If the fundamental schemas of the database are changed, which is more likely still to be correct syntax? And not least, which more eloquently says: *Come and be an astronomer for a night?*

6. Compare Kernighan and Plauger (1974): "If someone could understand your code when read aloud over the telephone, it's clear enough. If not, then it needs rewriting."

There is a database of a sort beneath any work of IF, too: the collection of rooms and items, with their properties and spatial relationships. Here, too, natural language is concise and expressive even when it contains only elementary grammar:

> East of the Garden is the Gazebo. Above is the Treehouse. A billiards table is in the Gazebo. On it is a trophy cup. A starting pistol is in the cup.

The combination of what is explicit and what is implicit in this 31-word source text is sufficient to compile an IF story file with three locations, one supporter, one container and one miscellaneous item (the starting pistol).

So much for the accusation that natural language code is necessarily verbose. Because natural language can be used ambiguously or sloppily, it is also sometimes dismissed as "imprecise", but this overlooks the fact that many important precise documents are written in natural language (standards documents, scientific papers, medical prescriptions), and that people are generally very exact in everyday conversation. Consider the following:

> A weight is a kind of value. 10kg specifies a weight. Everything has a weight. A thing usually has weight 1kg.
>
> A container has a weight called breaking strain. The breaking strain of a container is usually 50kg. Definition: A container is bursting if the total weight of things in it is greater than its breaking strain.
>
> A lead pig, a feather, a silver coin and a paper bag are in a room called the Metallurgy Workshop. The bag is a container with breaking strain 2kg. The lead pig has weight 4kg. The feather has weight 0kg.

Inform can now unambiguously test whether "a container held by someone is bursting". Such descriptions narrate complex circumstances with extraordinary clarity. Also of note here is the representation of constant weights: we write "2kg", rather than storing the number 2 in an integer variable, as we would in most conventional programs to achieve the above effect (perhaps using typechecking to distinguish weight-integers from other integers, if we can be bothered to be so pedantic). The code continues, and while it does begin to take the form of a procedural routine, there are only three instructions, each of which does something distinctive from the others:

```
Every turn when a container (called the sack) held by someone visible (called the
unlucky holder) is bursting:
    say "[The sack] splits and breaks under the weight! [if the player is the unlucky
    holder]You discard[otherwise][The unlucky holder] discards[end if] its ruined
    remains, looking miserably down at [the list of things in the sack] on the floor.";
    now all of the things in the sack are in the location;
    remove the sack from play.
```

The mathematician Paul Halmos once said that one should write papers as though one were explaining matters to a friend on a walk in the woods, with no blackboard or paper to hand: to use the fewest possible symbols and notation makes for clearer exposition. Here we indeed minimise names. The procedure needs none, as it does not need to be called (we just say when it happens), and there are only two variables ("the sack" and "the unlucky holder") and the five different loops implied by the code have no loop variables. In "a container held by someone visible" there are two potential searches (through containers

and through visible people), though in fact internally it is optimised into a single loop; "all of the things in the sack" again implies a loop, as does "the list of things in the sack", and "the total weight of things in it".

Many features of natural language are readily imitated in conventional code: a verb juxtaposing nouns is a procedure call with its arguments,[7] an adjective is a function which returns true or false, a proper noun ("Mr Jones") is a constant (or an object), a common noun ("man") is a data type (or a class), prepositional phrases ("Jack is in the box") could be regarded as tests of standard data structures such as trees, and so on. But Inform is considerably strengthened by two aspects of natural language less easily visible in orthodox programming: tenses and determiners.

If our aim is minimise the number of named variables, it is worth nothing that many variables in IF are essentially counters or flags, that is, totals or ways to remember past states of play: whether something has happened or not. In natural language, we can simply say "if the black door has been open" or "the number of things on the table", avoiding the need for flag or count variables, with their names to be remembered, and the possibility of error in initialising them.

> Instead of examining the tapestry for the third time, say "All right, so it's a masterpiece, but is this really the time to make a detailed study?"

Inform has four tenses: the present ("is"), the present perfect ("has been"), the past ("was") and the past perfect ("had been"), allowing us to discuss the situation in comparison with history either now (in the present) or at the point where the current action began (in the past).[8] Determiners, on the other hand, underlie Inform's ability to imply searches and loops. These are the words at the head of noun phrases which give them a degree of definiteness. Even the innocuous word "a" might imply considerable activity: "if a person is carrying a container" is a double search, over both people and containers. Natural language is rich in determiners: Inform allows, for instance, "not all of", "at most three", "almost all", "some", "most", and so on.

7. Though it must be admitted that frequently occurring English words have difficult meanings, and verbs are not as easily explicated as this casual mention suggests. The copular verb *to be* is especially hard to analyse, partly because of the pre-eminent role played by equality in predicate calculus, partly because of an etymological accident in early English which conflated together meanings distinguished in most other languages. Thus the word "is" will sometimes be compiled to the computer-programming operation "=" (set equal), sometimes to "==" (test if equal), sometimes "=~" (the Perl operator for pattern-matching) and sometimes to a kind of implication not generally found in programming languages. ("If all the green trees are tall" means: if green and tree implies tall then...) Similarly, "of" is used in about ten grammatically different ways in Inform.

8. Tenses nevertheless pose formidable implementation difficulties. Consider what we must do to put ourselves in a position to answer the question "has the President ever been ill?": clearly we need to maintain a continuous medical history in order, one day, to be able to look back over it and answer the question. But how often do we check the President's health: daily? weekly? annually? (Inform has a convenient answer here: it divides time naturally into actions, and performs such "maintain a history" checks between actions.) A more serious problem is that "has the President been ill before?" poses an ambiguity. "The President" is not a constant, but a variable: the post is held by different people at different times. Do we mean the person who is President at the "point of reference" of the tense – i.e., the time when we ask the question – or the "point of event" – the time when the illness occurred? (See Reichenbach, 1947, for this way of thinking about tenses.) Suppose the question is asked in 2005. If we substitute a value into the variable at the point of reference, the question resolves to "has George W. Bush ever been ill?"; if at the point of event, it becomes

Many of these points in favour of natural language programming would apply to a wide variety of situations (searching the Sloan Digital Sky Survey, for instance): but my belief that *the natural language for writing IF is natural language* is based ultimately on the special nature of interactive fiction. IF is based on a dialogue of text between reader (or "player") and computer, with both directions of communication prompted by textual possibilities supplied by the author. That means we have three agents describing the same situations — author, computer, player — and in an orthodox programming language such as Inform 6, the same idea accordingly has three different expressions. To specify a typical object, the author must specify all three of these: the source code constant `willow_pattern_vase`, the description text "willow pattern vase" and the parsing data `'willow' 'pattern' 'vase'` used to recognise the object in the player's typed commands. But words are just words, and it is repetitious and artificial to have to write them differently all three ways. A natural language description simply refers to "a willow pattern vase". It collapses the separation between author and player.

Similarly for states of things. If we wish a bottle to be empty, half-full or full then in most IF design systems we write code which stores (say) the number 1, 2 or 3 in some variable, then have to convert (say) 2 into the text "half-full" when printing out what is going on, and make the reverse conversion when parsing a command like "drop half-full vase" – a separation between author and player which obstructs the design process and increases the likelihood of bugs. It also results in IF code unnaturally full of numbers. Integers are so ubiquitous in computer programs that programmers easily forget how seldom they naturally arise in English text. Consider how the use of numbers (cloud 9, perhaps) to represent weather patterns would have obfuscated the following:

> "Weathering"
>
> A cloud pattern is a kind of value. The cloud patterns are cumulus, altocumulus, cumulonimbus, stratus, cirrus, nimbus, nimbostratus.
>
> The Mount Pisgah Station is a room. "The rocky peak of Mt. Pisgah (altitude 872m) is graced only by an automatic weather station. The clouds, close enough almost to touch, are [a random cloud pattern]. Temperature: [a random number from 7 to 17] degrees, barometric pressure: [950 + a random number from 0 to 15] millibars."

The use of square-bracketed substitutions in text also increases clarity. In Inform 6, for instance, the above room description would have required a routine to be written, expressing unnecessary intermediate steps (such as putting a random number into a

"has an incumbent President ever been ill during his term of office?" My own feeling is that substitution at the point of reference is more likely to be the natural reading of the question, but this is almost impossible to implement in a computer program. In this example, it would mean keeping medical histories for every person born in the USA, because in (say) 1974 we don't yet know that George W. Bush will be President in 2005: it could be anybody. The only practicable implementation is to carry out variable substitution at the point of event: to ensure that one will be able to answer "has an incumbent President ever been ill during his term of office?", one only needs to monitor a single person's health at any one time, which requires far less effort and record-keeping. The ambiguity is a genuine problem afflicting Inform's reading of past-tense conditions such as "if the noun has been open", where "the noun" is the object typed in the player's current command – and is therefore, like "the President", a variable in constant flux. Substitution at point of event means that past tense conditions relating to rapidly changing variables sometimes do not do what the user expects: which is a bad thing. Inform's documentation on past tenses strongly suggests that people use them with constants rather than variables.

variable) and so splintering the actual text that it would have been difficult to get a sense of its literary style by looking at the source code.

Inform does not aspire to recognise anything like the whole sweep of natural language, and in a few cases usefulness has been allowed to trump linguistic fidelity: in particular, it does not attempt to reject all *un*-natural language. But on the whole Inform tries to avoid eccentricity. The four self-imposed guidelines for the language were as follows:

> 1. A casual reader should easily be able to guess what a sentence does, and that guess should be correct.
>
> 2. The language should be economical, but not to the point where this compromises its intelligibility.
>
> 3. If in doubt as to syntax, imitate books or newspapers.
>
> 4. Contextual knowledge is best supplied by the author, rather than being built in.

Rules 1 and 2 are motivated partly by the basic aesthetics of natural language: the whole point was to be able to write a text, and a text should be legible. But there is another justification. One reason for COBOL's unexpected survival to the 21st century as a language for handling, say, financial transactions in the City of London, is that however much today's coders look down on COBOL as a verbose anachronism, they can still understand COBOL programs written in the 1970s and continuously used since: COBOL's priority of intelligibility over economy (rules 1 and 2 above) acts as something of a preservative against "code rusting". This is relevant to IF since IF authors – like novelists – are creating a cultural product to be accessible indefinitely, not a tool for immediate use and rapid disposal.

Rule 3 is best justified by the marks it has left on Inform's design. The most obvious imitation of printed books is that Inform projects have chapter headings, contents pages and an index (with Inform generating the contents and index automatically). Inform's equivalent of "printf" – the formatted printing command which every programming language needs – also imitates print culture:

```
say "You've been wandering around for [number of turns in words] turn[s] now."
```

The syntax here mirrors the journalist's rule that quoted matter in square brackets can be paraphrase rather than verbatim text. Escape characters are also eschewed when we want double quotation marks to indicate speech inside text which is already double-quoted: the convention is that single quotes should be used, which are automatically converted to double-quotes in printing. This follows the standard bibliographic convention on citing journal articles which contain quotation marks. Definitions of new adjectives are set out as they would be in science text-books:

```
Definition: Something is invisible if the player cannot see it.
```

The greatest prize from rule 3, however, was the solution to an awkward question: how could Inform cope with data structures such as arrays? Conventional arrays are clearly unviable in natural language, with their indexing subscripts (we are trying to abolish spurious number variables, after all). Searching and cross-referencing of arrays only makes constructions like AppleScript's "item 23 of..." syntax painfully obstructive. We need a way to create, discuss, look up, change and cross-reference arrays – ideally multi-

dimensional arrays, with functionality broadly equivalent to Perl's associative hashes – all without explicit subscripts and loops.

Printed books do, of course, include data structures. While they do sometimes lapse into diagrams, more often these data structures take the form of tables, typeset alongside the main text but given a title which the main text can refer to. Following rule 3, Inform does the same:

Table 2 - Selected Elements

Element	Symbol	Atomic number	Discovery
"Hydrogen"	"H"	1	a time
"Iron"	"Fe"	26	
"Zinc"	"Zn"	30	
"Uranium"	"U"	92	

Note that some columns are filled in, others left blank to be filled in during play (we shall suppose). We can look up data like so:

```
the atomic number corresponding to a symbol of "Fe" in the Table of Selected
Elements
```

(which is 26), or we can set conditions such as

```
if there is an atomic number of 51 in Table 2
```

(no). We can sort the table in various ways, have entire blank rows, and so on: we can even loop through the table, in what is unavoidably a procedural style of coding, without the use of a loop variable. Rows are selected in turn, rather as they are in scripting languages for databases.

```
repeat through the Table of Selected Elements in reverse symbol order begin;
    say "[symbol entry] is the international symbol for [element entry].";
end repeat.
```

Rule 4 of Inform's guiding principles, "Conceptual knowledge is best supplied by the author, rather than being built in", leads us into issues covered in subsequent sections of this paper. Briefly, though: what should an IF design system "know"?[9] The examples given so far demonstrate that Inform "knows" something, at least, about spatial arrangement. It will object if we say that a box is in the canoe and also that the canoe is in the box, which

9. There are perhaps three reasons why one would say that Inform does not "understand" English: (i) it is a computer program, which experiences neither the sensory world nor human society; (ii) without wishing to get into the dispute over whether human knowledge is more like a dictionary or an encyclopaedia, Inform has neither of these; and (iii) some of its mechanisms are so simplistic that we would certainly not ascribe human characteristics to them if we could see how they worked. On the other hand, Inform passes certain simple tests, such as the 1958 experiment in which it was shown that a five-year old child can, if told about a furry animal called a "wug", spontaneously use the plural "wugs" even though this is a word never heard before. So in our anxiety to insist that Inform has no real language ability and no real knowledge, we may be overlooking something. The ever-maximalist Jackendoff proposes that terms such as f-understand and f-mind should be used in place of understand and mind, where the f- prefix stands for "functional": that, in effect, a working mechanism to accomplish something may be said to f-understand its task by virtue of the fact that it works. To that extent, Inform does f-understand non-trivial semantic concepts – which is what I mean by saying that it "knows" about spatial arrangement – and the declaration in its manual that "Inform does not understand English" is simplistic.

it could hardly do without some functional knowledge of containment. Similarly, it knows that "four eggs" counts as "more than three eggs", and therefore has a functional knowledge of cardinal numbers: we could find many other examples, but Inform's ignorance of basic, shared human knowledge vastly exceeds its understanding. The following sentence arose in testing the second beta:

> The life support unit fits the egg.

This led to a bug because Inform construed the verb as "support" and not "fits", as a result creating items called "life" (which it guessed to be plural) and "unit fits the egg".

Why allow such ignorance: why have rule 4? One answer is that the alternative, a comprehensive contextual human knowledge, is far beyond the state of the art in artificial intelligence, but this is a cheap response: clearly Inform could get a lot nearer to the state of this art if its author made more of an effort. The real reasons are that the source text will be more self-explanatory if it takes less knowledge for granted, and that the language will be more flexible if it does not impose preconceived ideas. We shall see this latter point again in §2a when discussing taxonomy and the use of common vs. proper nouns.

Much of what is new in Inform 7 – the emphasis on rules, type-checking, actions which pay attention to success or failure, table searching – could have been achieved by incremental improvements of Inform 6, preserving most of the existing C-like syntax. They did not require the adoption of natural language, and nor did the new user interface (some of which in fact also works with Inform 6 projects). The general reaction of experienced IF writers to early drafts of Inform 7 was a two-stage scepticism. First: was this just syntactic sugar, that is, a verbose paraphrase of the same old code? (The cynical reader will have relished the lapse into "begin; ...; end repeat" above: iteration, and table searching, are generally responsible for the least "naturally" legible Inform 7 source text.) Second: perhaps this was indeed a fast prototyping tool for setting up the map and the objects, but would it not then grind into useless inflexibility when it came to coding up innovative behaviour – in fact, would it be fun for beginners but useless to the real task at hand? It sometimes seemed to those of us working on Inform that an experienced IF author, shown Inform 7 for the first time, would go through the so-called Five Stages of Grief (Kübler-Ross 1969): Denial, Anger, Bargaining, Depression, and Acceptance. The following comment is typical of the Bargaining stage:

> I would like to see it be as easy as possible to mix Inform 6 and Inform 7 code. [...] I also wonder if it might be possible to allow the user access to the Inform 6 code that the Inform 7 pre-processor creates. I can imagine some people wanting to use Inform 7 to lay out the outline of their game – rooms, basic objects therein, and so on – quickly, and then do the heavy lifting, so to speak, in Inform 6.

In fact, Inform 7 *does* allow the inclusion of Inform 6 excerpts, but in an effort to conceal this from the user, the ability is not mentioned in the manual until the final, intentionally not-for-beginners chapter: after some 260 examples of natural language doing "heavy lifting" have, with any luck, eased the reader's passage through Depression to Acceptance.

Whether natural language will be widely accepted by the IF community, time will tell. Certainly the legibility of Inform source text depends very much on the willingness of the author to cooperate: sometimes being willing to type more text in order to choose names

which make grammatical sense, for instance. It is also sometimes easy (and harmlessly amusing) to fall into double-entendres: "Men are usually transparent" and "A god is a kind of value" are genuine examples from Inform's beta-testing.[10] Sentences may look natural and yet be false friends, as translators say of words in foreign languages which look close to English ones but have different meanings: *actuellement* is not the French for *actually*. Whereas I myself would spare no effort to avoid unnatural source text, Inform being my baby, other testers were more interested in getting something done than in how the source looked. But the testing team are perhaps not representative of the system's eventual users, who will be able to learn a more stable language.

Extensive work has been done on natural language recognition in computing, and Inform makes no claim to originality in its inner workings: indeed, it could be regarded as a descendant of Winograd's program SHRDLU. (For a discussion of SHRDLU in the context of IF, see Montfort (2004).) The one practical lesson I would like to record here is that the biggest source of bugs in Inform came from my own imprecise knowledge of grammar. For instance, if we suppose that open and closed are antonyms, should "the door is not closed" be read as equating "the door" and "open", or as forbidding the equation of "the door" with "closed"? The result will be the same, but the second way is a better implementation, because it is more consistent with other verbs: the placement of "not" after the verb is almost unique to the copular form of *to be*, and its apparent association with the noun which follows is only a quirk of irregular English usage. We are likely to write cleaner code if we implement "is not" in the same way as "does not carry": and we will not end up accidentally parsing "does not carry not".

10. Men are no longer transparent: this was a leftover from Inform 6's quaint ideas about "concealment", which treated animate and inanimate holders the same, thus making it difficult to express the idea that someone is *purposefully* hiding something. Inform 7 eventually recognised this by focusing on the intention of the holder, not the state of the thing held.

§1c. The primacy of rules over objects

The third of the three fundamental changes in Inform 7 as compared with traditional IF-design systems, after the user interface and the adoption of natural language, is the replacement of an object-oriented model by a rule-oriented model.

The successful IF design systems to date[11] have mostly been object-oriented forms of C, with one important exception: Infocom's proprietory compiler ZIL, an object-oriented form of LISP. This happened partly because technology, 1975-95, obliged the source language to be efficiently compilable. (Until about 1990, an IF story file would be larger than the memory capacity of the computer which compiled it.) Those who wrote IF compilers tended to write them in C, were therefore fluent in C, and saw an IF-specialised form of C as an ideal system to use. The use of an object-oriented language with a strong class hierarchy was orthodox to the computer science of the day, and it neatly resembled the "hardware" of IF – the standard data structures used in story files, as exemplified by Infocom's Z-machine. In this way 1970s implementations of IF were taken to be the structural model for IF itself. I contend that, on the contrary, IF is not best described as object-oriented but as rule-oriented.

I concede that bundling properties together into object and class definitions, with inheritance from classes to instances, works well. My objection is rather to the doctrine that when components of a program interact, there is a clear server-client paradigm; that one component exists to serve the needs of another. The contents of a work of interactive fiction are typically not in such relationships. If facts concerning a tortoise must all be in one place, facts concerning an arrow all in another, how are the two to meet? It seems unnatural to have a tortoise-arrow protocol, establishing mutual obligations. Neither exists to serve the other. The tortoise also eats lettuce, meanders about garden locations and hibernates. The arrow also knocks a flower-pot off a wall. By the same token, the world of a large work of interactive fiction is a world of unintended consequences. New and hitherto unplanned relationships between components of the "program" are added in beta-testing, something which the programmers of, say, a spreadsheet would not expect.

A second objection is that object-oriented code for IF divides into two quite different blocks of material: (1) the objects and classes, with their properties and specific behaviours, containing the materials for the game; and (2) the mass of usually procedural code containing the standard mechanics of play – the "library", as Inform calls it (Infocom sometimes used the term "substrate", perhaps a better image). This disjunction between specific rules (1) and general rules (2) is problematic for both. First, the general rules are hard to change, because although the language is rich in flexible ways to modify specific facts (object properties), only the crudest mechanisms exist to modify general facts (library routines). The specific rules have the opposite problem: specific rules are easy to apply but the realistic constraints of the world model are often lost in the process, because "realism" is the province of the general rules which they pre-empt. For example: suppose a golden

11. Fighting talk. I really mean "the successful IF programming languages", since design systems wiring objects together for processing by a standard run-time engine – such as Scott Adams's Adventure International system, The Quill, AGT, or ADRIFT today – have at various times proved fruitful and popular. The rise of ADRIFT demonstrates that "constructor kits" have, in fact, made a comeback.

apple is inside a closed transparent box. General rules about impermeability forbid the taking of the apple, but here we shall allow the player access provided he is wearing a magic ring. How is this to be done? It seems inappropriate for such a one-off circumstance to be spatchcocked into general rules. So we should write a specific rule instead, but that means attaching behaviour to a specific action: the taking of the apple. In Inform 6, we might modify the apple as follows:

```
before [;
    Take: if (apple in box && ring has worn) {
    move apple to player;
    "Your hand passes through the glass to grasp the golden apple.";
    }
],
```

And this is also unsatisfactory. Whether we regard it as behaviour of the box, or of the magic ring, it is certainly not a behaviour of the apple. What if the player tries to put the apple back again, or to put something else inside? Or turns the apple instead of taking it? What if the player is only allowed to carry four items, and is already fully laden? What if the transparent box is inside a locked cage to which the player has no means of entry?

The traditional solution is to rewrite the general rules to include yet another hook on which customised code can be hung. The accumulation of such hooks makes IF design systems grow steadily more complex as they age, and no matter what is added, it is never enough. As Andrew Plotkin observed to me, when early drafts of Inform 7 had reached this same impasse: "I'm tired of not being able to override some behavior, because there's no hook there. I want the world to be made of hooks."

Making the world of hooks, indeed. Where Inform 6 would use a central chunk of code with a few hooks attached ad-hoc, Inform 7 uses a line of hooks with pieces of code attached ad-hoc. These pieces are conceptually individual rules, and each is named. The "rulebooks" (these lines of hooks) are highly modifiable, even during play. Special "procedural rules" take precedence, like points of order in debates:

> Procedural rule when the ring is worn: ignore the can't reach inside closed containers rule.

Of course it is still true that some rules are narrowly specific and others widely general, but many lie in between and there is no longer a categorical distinction but rather a sort of stratification. More specific rules take precedence over less specific ones, and the order in which the source code defines them is (mostly) irrelevant. This has two profound consequences: that the author can arrange the source as he pleases, choosing for instance keeping everything relevant to particular events or scenes together; and that four or five different sets of extension rules, by different authors, can peaceably coexist in the same work of IF, even though they all modify the same original definitions.

A system of gradations of rules has two prerequisites. First, the compiler needs a solid understanding of types: in particular, a way to judge whether one category ("an open container") is a special case of another ("a container"). Secondly, the author needs a flexible way to describe the circumstances in which rules are to apply. Both considerations mesh well with the use of natural language: to make sense of the ambiguities of English, a strong typing system is needed anyway; and natural language is good at describing circumstances ("in the presence of Mrs Dalloway") and categories ("a woman in a lighted room").

Part 2. Naturality in Theory

§2a. Semantic analysis: a sampling of case studies

It seems to me that the most profound difficulties in natural language recognition lie not in computing, where somewhat complacent textbooks exist to demonstrate standard algorithms (for instance, Allen (1995)), nor even in the arcane and by no means settled post-Chomskian field of formal syntax (Culicover (1997); Culicover and Jackendoff (2005)), but in the field of semantics. Definitions of semantic analysis vary. Some, such as Ray Jackendoff (2002, §9.1), stop little short of defining it as the theory of human thinking:

> If you are not prepared to deal with at least language, intelligence, consciousness, the self, and social and cultural interaction, you are not going to understand meaning.

A more measured answer given by Davis and Gillon (2004) is that semantics is concerned with two basic questions:

> What values are to be associated with the basic expressions of a language? How does the meaning of the simpler expressions contribute to the meaning of the complex expressions the simpler ones make up?

The source text for an interactive fiction is, however, more directly referential than most texts. Contrary to Fenollosa's observation that there are, in reality, no nouns in the universe, the universe of an IF contains nothing but nouns, and the word "stone" is in a sense a stone. A program such as Inform has an easy life compared with a human reader: it may work by forming a model out of its source text, but it does not have to embed that into some greater model of an "outside world". It has the further luxury of reading text guaranteed to be truthful, literal and addressed directly to itself. Even so, most of the hard issues in semantics seem to arise in at least a toy form. For instance, Inform reads sentences quite similar to the notorious "donkey anaphora" example

> If Pedro owns a donkey he beats it.

in which the rules governing "it" have been construed in an astonishing variety of ways by linguists, none wholly convincing. Does "it" substitute for a specific thing (Chiquita, Pedro's hypothetical donkey), making it a deictic pronoun, or for a universal ("every donkey such that..."), making it a bound pronoun? Or is our thought process not analogous to putting a value into a variable at all?[12]

Semantic analysis falls into two strands. Some linguists regard sentences as logical propositions, while others regard them as repositories of shared concepts, unspoken ideas and sense-perceptions: see Goddard (1998) for a fascinating account of, for instance, colours and emotions as expressed in different world languages, and for summaries of a number

12. Since this question was first raised by Geach in 1962, the literature of semantics has been routinely cruel to donkeys in the most callously offhand fashion. If they had a clubhouse, the motto over the door would be "Every farmer who owns a donkey beats it." But before you dismiss the anaphoric pronoun problem as trivial, try to explain why you didn't read "they" in the last sentence as referring to the donkeys, even though it is the only plural noun phrase in this footnote.

of provocative views on how meaning may be composed. Jackendoff uses the analogy of 18th-century chemists dimly glimpsing a Periodic Table of the elements: but others prefer a comparison with the Linnaean botanists indefatigably sorting plants into species, families and so forth, until they in some sense discover a hierarchy of meaning.[13] Either way, writers on semantics often seem to regard their field as being in its infancy, which is sobering when we consider that the first analytic grammar was written in cuneiform by a Babylonian.

There is no space here for a detailed discussion, nor should I pretend expertise in the field, but perhaps a selection of case studies relevant to IF may serve to support my general contention that semantic analysis is of central relevance to a theory of interactive fiction.

Q1. *Does IF overstress hierarchies of containment?*

Inform, in common with all traditional systems for IF, has spatial ideas hard-wired into it: in particular the use of a tree structure rather than a "flat" map to model the location of things. It would be disquieting to think that we do this simply because it is easy to implement tree structures on computers. I therefore draw comfort from the number of linguists who also regard CONTAINER as a central idea: that it is, for instance, an "image schema" by which we metaphorically extend our bodily ideas of inside and out onto the world around us. The body being a container of great importance to us, we correspondingly picture the spatial world around us in terms of containment even when it is seldom so clear-cut. Why do we say that we are "in bed", for instance, when we seem to mean that we are on top of something and are more than half covered by a blanket, and when we would probably say "no" if a child asked us whether the blanket is part of the bed? Similarly, if we are asked "Where are the teaspoons?" we are less likely to answer "In the kitchen" if they are not in plain view: we will say "In the kitchen drawer." If A is "in" B which is "in" C, we are reluctant to deduce that A is "in" C. (For this reason, "in" is not a transitive relation in Inform.) On the whole I think the literature of semantics offers some evidence that IF's hallowed use of an object tree is genuinely natural.

Q2. *Do objects in IF really have kinds, and what for?*

Suppose we start from the premiss that an IF design system like Inform will have to produce a computer program which contains a wide range of different data structures, from the number 1815 to a packet of bits which in some sense models the character of Napoleon Bonaparte. This range is miscellaneous both in (i) the nature of the underlying ideas (numbers are not like men) and (ii) their binary representation in the final program, and if our compiler is doing any kind of sensible job, then these miscellanies will coincide: that is, the compiler will choose suitable forms of binary representation (ii) on the basis of the semantic concepts we seem to need (i). So much is true of any present-day compiler, gcc for instance, and it argues that an idea must exist at least internally of "kind" or "class" or "type". (For most programming languages, C for instance, this idea is explicit in the

13. These analogies, pleasing as they may be, are not necessarily good ones. The Periodic Table is a relatively rare case in nature where fundamental science about simple objects in isolation leads to clear-cut distinctions and affinities. The human world does not resemble this picture. And Linnaeus intended his taxonomy to cover all concepts in existence, not merely biological species: but his project essentially failed in domains outside biology, and even there has frequently been challenged.

source text, but I think we could conceive of an apparently typeless language where the idea was concealed.) We now move on to observe that since Inform is to take a piece of natural language as its source text, all such data structures are to be referred to by noun phrases: thus *1815* and *Napoleon Bonaparte*. This gives us a third miscellany: (iii) the range of possible meanings of noun phrases, and once again I argue that if Inform is behaving anything like sensibly, this must at least broadly coincide with (i) and (ii). In short, Inform's objects have to have kinds because data structures do in any computer program, and also because these particular data structures marry up to noun phrases in English, meanings of which also fall into kinds – such as NUMBER and MAN.

We seem now to have chased back to human practices, so we might ask: why do human beings use "kinds"? A concise answer is given by Taylor (1989): "the function of categorization is to reduce the complexity of the environment".

But I think we should distinguish between complexity of recognition and complexity of description. A child's set of kinds need to equip him to tell things which move by themselves from things which don't, or to tell food from furniture: to cope with whatever presents itself next in an ever-expanding world, which is primarily a problem of *recognition*. Inform is interested instead in achieving the simplest written expression of single, fully-known situation in the imagination of the writer: a problem of *description*. A human being unable to categorize food would be ill-equipped for life in any conceivable culture, but in a written work which happens not to involve any eating, there would be no need for such concepts.

In Inform it is a foundational principle that every thing has a "kind": and the function of these kinds is reduce the complexity of description. But as Borges reminds us in his spoof article on the subject, dividing up the world into kinds is not so easy as it looks:

> These ambiguities, redundancies and deficiencies remind us of those which doctor Franz Kuhn attributes to a certain Chinese encyclopaedia entitled 'Celestial Empire of benevolent Knowledge'. In its remote pages it is written that the animals are divided into: (a) belonging to the emperor, (b) embalmed, (c) tame, (d) sucking pigs, (e) sirens, (f) fabulous, (g) stray dogs, (h) included in the present classification, (i) frenzied, (j) innumerable, (k) drawn with a very fine camelhair brush, (l) et cetera, (m) having just broken the water pitcher, (n) that from a long way off look like flies.

And we must therefore go on to further questions about kinds.

Q3. *Are kinds immutable during play?*

In Inform, CONTAINER is a kind. As with an integer in a typical computer program, if something is a container at the start of play, it will always be a container throughout. But is this only a convenience of computing? I think not. The property of being open may come or go, but we humanly expect a container to remain a container, and when a physical object is so treated that it absolutely loses its kind – for instance, a house being demolished, or a potato mashed – we tend to regard such an event as a dramatic change in which, in effect, one object is replaced by another which has little in common with the original.[14]

14. Just as, in typical IF situations, an object representing a glass bottle may be withdrawn from play when the bottle is smashed, and an object representing broken glass brought into play to substitute for it.

Q4. *Can something be only partly of a given kind?*

Inform, in common with other IF systems, provides a certain stock of kinds and makes every object belong to (at least one of) them. But inevitably, IF writers find that what they want does not exactly fit into any of the kinds provided for. They sometimes find themselves stripping away all the behaviours of the kind of something, but being unable to actually change the kind, an unedifying spectacle. Is a CONTAINER which cannot contain still a CONTAINER? Perhaps not. It depends what you call a "container". From the point of view of IF, I suggest that the most persuasive way to define the meaning of a kind is to imitate a style of conceptual analysis proposed by Anna Wierzbicka in the 1980s. For instance, it is posited that the meaning of BIRD is that it is a "kind of CREATURE" with a collection of what might be called default expectations, none certainties:

> people say things like this about creatures of this kind:
> they have feathers
> they lay eggs
> they can fly

The suggestion is that any of these properties can be overridden in particular cases without necessarily stopping the creature from being a BIRD: thus, FLIGHTLESS BIRD is not a contradiction in terms, but is a "kind of BIRD" about which people say that "they cannot fly". This is exactly how rules are made about kinds in Inform, only to be contradicted by rules about specific instances of those kinds: see §1c above.

Q5. *Should Inform have many or few kinds?*

Inform has very few kinds "built in": indeed, dramatically fewer than rival systems,[15] and this takes some explaining away. Let us proceed from the answer to Q2: the function of kinds in Inform is to reduce the complexity of description of what the writer imagines.

Is description made less complex by having many kinds, or few? There are approaches to semantics which want to make pretty well everything a kind, in order to make sense of determiners and proper vs. common nouns: for instance, "A registered nurse is in Piccadilly Circus" involves the kind "registered nurse" being made actual (or "given an indexical feature") by the determiner "A"; more contentiously, we can go the other way, since any proper noun like "Piccadilly Circus" becomes a kind just by writing "a kind of Piccadilly Circus", or (say) "Imagine another Piccadilly Circus, but without the traffic". Perhaps analogy works this way: "It's like Piccadilly Circus in here." Inform does not accept any of this: to Inform, a kind is not a meaning. Rather, kinds exist so that we can make (or

15. Compare TADS 3, which after many years of thought (led perhaps by David M. Baggett in his work on the "WorldClass" library) currently has around 420 main classes, varying from generative-semantics fundamentals such as SensoryEmanation to highly specific gizmos: Candle, AutoClosingDoor, ShipboardDirection. This is not meant as a criticism, merely an observation that a rule-based natural language system may be most useful if it minimises the number of basic meanings, whereas an object-oriented programming language may serve its users better by going the reverse way. It should also be noted that TADS 3 does not distinguish between concrete and abstract objects to the extent that Inform 7 does (again, not a criticism: I increasingly suspect this is "right"), and that this partly accounts for the profusion of classes.

question) generalisations about them. We will certainly not make "Piccadilly Circus" a kind, nor even "a kind of Piccadilly Circus", and Inform can read "a registered nurse" as *either* determiner + kind (if the source text has already established that registered nurses are a general feature of the world) *or* as a one-off physical object which is not an instance of some more general phenomenon. Do we lose anything by not insisting that every noun phrase is headed by a kind? I think perhaps we do, but that we overcome it with only a little verbiage. For instance, the following is unambiguous to a human reader:

> if a woman is in a room, say "[The woman] is in [the room]."

but Inform does not allow it, because it does not freely convert kinds back into instances through some theory of the determiner "the" adding an indexical feature to a meaning ("woman", or "room"). Inform therefore obliges us to use a circumlocution like this:

> if a woman (called the inhabitant) is in a room (called the place), say "[The inhabitant] is in [the place]."

We therefore reject one, impractically maximalist answer to the question: not every noun phrase is going to be a kind in Inform.

Instead, a kind has to be a widely applicable meaning, about which we can usefully lay down general laws. This precept immediately shows that some even apparently sensible concepts would make bad kinds. For example: JADE is arguably a bad kind in science because it combines things with no underlying unity. "Jade dissolves in acid" is a dubious statement and "the density of jade" a meaningless idea, since there happen to be two chemically unrelated minerals which accidentally look alike and are both called "jade" (see Fodor 1998). Or again, LEFT-HANDED MAN is arguably a bad kind to have when describing medicine, because it misleadingly over-specialises. It is true and statistically demonstrable that "smoking causes cancer in left-handed men", but this statement might cause the reader to imagine that right-handed men are immune from lung cancer, perhaps due to holding their cigarettes differently. We cannot escape the importance of context here. In the world of cricket, LEFT-HANDED MAN is a very useful concept, and in my own life, JADE is a perfectly good kind since I am not a mineralogist and do not care whether the only jade ornament in my house is jadeite or nephrite. The point about context is important: different works of IF will need different sets of kinds. Inform must therefore be careful to define only those kinds which would make sense in every work of IF, and allow the writer of each individual work to define further kinds for himself. Of course, this is also necessary on grounds of practicality, but I argue that it would still be necessary even if armies of programmers were available to create a larger and larger IF design system.

So are there many, or few, kinds which would make sense in every work of IF?

One distinction made by linguists about kinds of physical object is between "basic" and "superordinate" kinds. It is mostly[16] common ground that kinds form a hierarchy, even though it is probably wise to keep an open mind about what form this hierarchy takes.

16. In the mid-1970s, Rosch conducted a series of experiments into what people consider "typical" objects: for instance, a chair is more typical of "furniture" than a music stand. Many different morals have been drawn from this about why chairs, say, are "typical" (they have a more basic purpose? they are physically simple? we use them more often? they exist in a typical house in greater numbers? they have no ancillary purposes? our parents teach us the word earlier? they are referred to by a phonetically simple one-syllable word?), and Rosch's results do indeed give those who believe in a taxonomic hierarchy something of an uneasy feeling. Still, CHAIR is a kind of FURNITURE is a kind of ARTEFACT is a kind of THING: no?

We can say with reasonable confidence, for instance, that TABLE is below FURNITURE. The idea behind the basic/superordinate distinction is that certain kinds are basic because they represent the natural answers people give to the question "What's that?" For instance, "A chair" is a more likely answer to the question "What's that?" than "A piece of furniture", so that CHAIR is arguably a basic kind, and FURNITURE is superordinate (which simply means, higher than basic). I am a little sceptical of this theory: to a Martian, I might indeed say "That's furniture", and in a chair factory I might say "That's a swivel chair". The set of kinds useful to a simple description of a situation inescapably depends on what the situation is. But to the extent that humans are more likely to be talking to other humans and more likely to be in domestic settings, perhaps there are kinds which are often "basic", like the pictures in a child's first book. So let us accept this basic vs. superordinate hypothesis. This immediately invites another rather large solution to our problem: not as big as "every noun phrase should be a kind in Inform", but still pretty huge – "every basic kind of physical object should be a kind in Inform". Inform should have a kind for CHAIR, for instance.

Again, Inform is able to reject this answer because it is used to describe only narrow parts of the world at once, and never needs a framework into which the whole world can be placed. Inform's strategy is to provide the fewest possible kinds – all of them superordinate, with perhaps one exception[17] – and to allow the writer to create whatever basic kinds are useful for the specific work being written at the moment. In a work of IF about chair manufacture, the writer is free to create kinds for FURNITURE, CHAIR and SWIVEL CHAIR: but Inform provides none of these in the core rules present in every work. Indeed, the core rules provide only 13 kinds of physical thing, one of these being THING itself.

This minimalist approach has many advantages: most obviously that it is feasible at all, without sidetracking us into attempting some completist simulation of all possible worlds. But it should also be said that a "high-up superordinates only" approach to kinds has its drawbacks for an IF design system using natural language as syntax, as Inform does. One reason Inform's core rules provide no CLOTHING kind is that the noun "clothing" is defective in English by not being countable. We cannot say "Two clothings are in the basket", and even "A shirt is a kind of clothing" looks odd: "A shirt is an item of clothing", maybe. While we could squirm out of this by substituting the word "garment", that is not ideal either. (The same issue arises in one of Inform's examples where a kind is created for ARMOUR.)

Q6. *Do kinds have overlaps? Can something belong to two kinds at once?*

A substantial change between Inform 6 and Inform 7 is that the new system is single-inheritance, that is, each thing can only have one immediate kind, whereas individual Inform 6 objects could belong to more than one class. While a thing can have more than one kind, this can only happen if kinds entirely contain each other: thus King Edward VII is a kind of MAN which is a kind of PERSON which is a kind of THING.

The corresponding question in linguistics would be whether, say, "a statue of Edward VII" belongs in any convincing way to a single kind. Is it a bronze pillar which happens, as a minor detail, to resemble a man; is it a man without animation; or does it belong in

17. The somewhat suspect kind "player-character", which is used to denote human beings from whose perspective the world can be explored. This kind exists largely for implementation reasons, and fits badly into the kind hierarchy. It may yet be abolished.

some sort of overlap of the outer fringes of PILLAR and MAN? Arguably natural-language meanings are neither singly nor multiply inherited from kinds, but are instead partially inherited ("a bronze pillar with some of the aspects of being a man" might define the statue reasonably well). This seems to me an exceptionally difficult question, and my inability to give a confident answer is a further reason why Inform is chary of defining too many built-in kinds: Inform takes the risk that we can, in fact, naturally represent the world with a strict hierarchy of kinds. (It mitigates this risk also by the flexibility it offers in attaching rules to selections of things on many bases other than kind alone.) There may well be alternatives to Inform's assumption that kinds do not overlap, but they are not likely to be simple.

Q7. *Are kinds of physical things like kinds of abstract concepts?*

An occasional, but persistent, trend in the development of Inform 7 has been that users want to use what Inform calls kinds of value (NUMBER, TIME, and so on) in the same way as kinds of object (MAN, DEVICE and so on), often expecting that syntaxes typical of one must necessarily be valid for the other, and reporting it as a bug rather than a feature request if not. (Particularly troublesome was SCENE, a chronological extent, not behaving like REGION, a geographical extent.) The source code of the Inform 7 compiler would probably be cleaner if kinds of value and kinds of object were handled with greater commonality, and it may be true that further integration of these ideas is possible, but at present I still believe that they are different in nature. This is not because of the superficial difference that NUMBER has an infinite number of instances, but there are only finitely many MEN. Rather I would point to the discussion in Jackendoff (2002) at §11.4, and to an interesting point about what information passes up and down the hierarchy of kinds.

In this passage, Jackendoff quibbles with taxonomies by suggesting that the usual idea of properties being inherited from above – for instance, a CHAIR inherits the properties of FURNITURE – may be too slow, or unnatural, or inefficient on storage requirements, to make this the true cognitive picture of how people guess the behaviour of things. (Maybe he has a point, but it seems to work for Inform.) But he then comes to an interesting point about a sort of reverse inheritance:

> More controversial perhaps is the question of whether a lexical concept carries structure relating it to *lower* members in the taxonomy. For instance, does TREE carry a list of its subkinds, including PALM, PINE, and PLUM? To carry things to an extreme, does PHYSICAL OBJECT carry within it a list of all its known subkinds? Implausible. On the other hand, people do use information derived by going down the hierarchy in order to draw inferences: this is "case-based reasoning". Of course, case-based reasoning is notoriously unreliable [...] but people use it all the time nevertheless.

By "case-based reasoning", Jackendoff means something like: "Mr Jenkins, my English teacher, was always bossy at parties. So that's teachers for you." – unreliably arguing, that is, from particular cases. In Inform, the distinction I would draw between abstract and physical kinds is that a "kind of value" does contain information about its known instances when it is created, so that case-based reasoning is valid: but a "kind of object" does not, even though it must only have a finite number of instances in any actual work of IF. I think inheritance may work differently in the abstract and physical hierarchies.

Q8. *Why are IF design systems good at modelling objects such as chairs or rocks, but bad at modelling substances such as water?*

Since Jesperson (1909) linguists have divided nouns into "count nouns" (chair or rock) and "mass nouns" (water or dough), and it is apparent that these two categories of noun have different semantics: two oranges, some bread, but not two breads, some orange. Enquiries into the meanings of mass nouns involve the meaning of "is a part of": one would probably not say that the left hand side of a loaf of bread "is a part of" the loaf, and certainly Inform's implementation of parts makes sense for count nouns but not mass nouns. If we ask what mass nouns such as rope, sand, fire and water have in common, indeed what makes them so troublesome for designers of IF, perhaps it is the problem that "part of" the thing is in one state, part in another.

One approach is to reduce the mass to individually counted atoms: say, to regard shingle as a pile of a hundred pebbles, or to regard rope as a chain of segments, or water as a collection of scattered puddles. This is sometimes manageable (ropes are often implemented as a chain of just two segments – the two ends) but can be inefficient and impractical at run-time when the number of atoms grows large. Most systems for liquids give up when they are divided too far. Another approach is provided by Inform's existing BACKDROP kind, which – for a multiply-present thing like, say, the moon or grass, found in many locations at once – is already a primitive sort of mass noun: this works nicely but only with the huge assumption that the substance is immobile and is present either once in any given location or not at all. As a result, it is indivisible, and one cannot pick the grass. A further issue is that, whatever method is used, mass nouns impose formidable problems to the run-time parser, which must allow the player to refer to one part of something as distinct from another.

It might therefore be said that these difficult things should be left to ad-hoc implementations by the programmer, and that no general provision should be made for them. But the importance of mass nouns in semantics suggests that they are important in conceptual pictures of the world. This argues, first, that Inform ought to provide natural ways to deal with them; and second, that the semantics of mass nouns may tell us what these natural ways are. At the moment I incline to the "single multiply-present object" strategy, rather than the "break up into atoms" strategy, since it seems more linguistically natural – all of the properties of a given puddle except for its location are, in fact, properties of water. It is also interesting to note Jackendoff's suggestion that count nouns are to mass nouns as areas are to places, or as processes are to events.[18] Inform has made less progress here than had been expected, but I hope to work further on this.

Q9. *How many nouns should an action involve, and can they be optional?*

It seems to be the case that verb phrases in natural language, such as may describe actions, take between 1 and 4 "arguments": extreme cases would be "Helen sighed" and "Indigo

18. Inform's index of scenes is indeed meant to be a "map of time", and is presented alongside the map of rooms laid out in space: Inform tries to treat time as like space and vice versa if possible, since natural language seems to do this, too. Inform's implementations of REGION and SCENE both involve single objects with many relationships: perhaps this is further evidence for the "single multiply-present object" strategy?

bought a jumper from Kevin for £10". Jackendoff (2002) comments that the 4-argument cases generally involve exchange between two agents – purchases, or wagers. Inform handles only 1 to 3 arguments in actions with any convenience, the subject ("actor") and up to two objects, so it is reassuring to think that a 4th argument is needed only in describing actions which require the simultaneous consent of two people: actions in systems for IF are impulses felt by a single person, and transactions are better regarded as a sequence of impulses (Indigo gives £10 to Kevin – 3 arguments; Kevin gives the jumper to Indigo – 3 arguments again).[19]

Optional semantic arguments are more problematic. Do they exist? For instance, "Lucy listened" and "Lucy listened to the machine" might be construed as: (i) one action with an optional argument; (ii) one action with a compulsory argument, but which is implicit in the first example and explicit in the second; or (iii) two different actions altogether, one a sort of ambient listening, the other a narrowed-down auditory enquiry. This has long been awkward for Inform: Inform 6 chose answer (i), but Inform 7 plumps for (ii), so that the bare command "listen" is implicitly read as "listen to the current location". I also half-believe (iii): but then, as Jackendoff points out, the examples of "Neil ate" (a valid sentence) and "Olive devoured" (not so good) suggest that the whole phenomenon of apparently-optional arguments may be more to do with lexical quirks — customary differences in usage between the words "eat" and "devour" — than any deeper semantic structure. If so, this justifies the stance taken by Inform 7 that at the semantic level all actions have a definite number of arguments.

19. Gregor Hohpe's wittily written piece 'Starbuck's Does Not Use Two-Phase Commit', a blog posting from 2004 reprinted in Spolsky (2005), argues that computing algorithms which construe exchanges as a simultaneous act by two parties are (a) not like real-life transactions, such as buying coffee in Starbuck's, and (b) often inefficient, compared to Starbuck's. Starbuck's does not use 4-argument verb phrases.

§2b. Predicates and quantification

Whereas §2a was concerned with what might be learned from big-picture semantics – in particular, the ideas of categorization which underlie human expression – much may also be gleaned from more basic linguistics, "down" at the sentence and even clause level.

It is a commonplace in most rigorous approaches to semantics that, whatever other cloudy ideas may hover around a sentence, its basic meaning can be transcribed as a proposition in predicate calculus. (See the introduction to Davis and Gillon (2004); for the mathematical background, see for example Johnstone (1987).) It would be wrong to suggest that there is any one agreed logical framework which acts as a sort of *ur*-language: elaborately different forms of predicate calculus have been proposed, as have a similar variety of methods for converting natural language into logical propositions.[20] But there is a strong consensus that a large part of the meaning of simple sentences can faithfully be transcribed into mathematical logic, and that this can be done by applying some mechanistic algorithm to the original text. (This is certainly what Inform, and almost every other text-recognition program, does.) Predicate logic has certain computational advantages, but its benefits are more profound. It gives us a rigorous framework in which to make deductions, and thus to infer implied information from text. For instance:

"somebody is in an adjacent room"
i.e., Exists x : *person*(x) and Exists y : *room*(y) and *adjacent*(y) and *in*(x, y)
i.e., Exists x : *person*(x) and *room*(*parent*(x)) and *adjacent*(*parent*(x))

where Inform was able to eliminate the second of the two unknowns (the room) by noticing that it depended directly on the first (the person), the elimination being made by a substitution:

y = *parent*(x)

The simplified form of the proposition is equivalent to the original, but compiles to more efficient code. A second advantage of predicate logic is that it leads naturally to model theory, another mathematical methodology useful to "understanding" a text. Given a long run of statements about some situation – a novel, for instance – how are we to form a picture of what is going on? Inform's usage of model theory will be discussed in §2c, but for purposes of the present discussion, I claim only that the project of comparing natural language with predicate logic has been very successful in linguistics. This suggests a close affinity between the two: so what is important in making predicate logic flexible may also be what is important in making natural language flexible.

Predicate calculus has various ingredients. If we consider a logical statement such as

Exists x : *person*(x) and *room*(*parent*(x)) and *adjacent*(*parent*(x))

the first thing we see is a quantifier: "Exists x" means that there is some value of x for which what follows is true. x is required to be a person, and to be in a room, which is

20. Happily, Inform works in domains where most of the real difficulties do not arise. For instance it needs to understand "if", which is easily reducible to mathematical logic, but not "because", which is not. Consider the variety of meanings possible in sentences which take the form "Plants grow because X", for instance.

moreover an adjacent room: thus, "somebody is in an adjacent room". "Somebody" is an example of a determiner in grammatical terms, or at any rate the "some-" part of it is. Standard predicate calculus has only two quantifiers, "For all" and "Exists", which is fine for "every man is mortal" or "some man is mortal", but is insufficient for Inform, which supplements them with Mostowski's generalized cardinality quantifiers ("Most", "At-Least-3", etc.) along the lines suggested by Barwise and Cooper (1981). Thus "if three people are angry" becomes

At-Least-3 x : *person*(x) and *angry*(x)

Just as generalized quantifiers expand the scope of predicate logic, allowing a wide range of determiners makes natural language more expressive. This is one example of how Inform became richer through comparison with predicate logic: to put it crudely, because quantifiers are important in logic, it seemed worth investigating whether Inform would gain from expanding its range of determiners. This step was taken about half-way through Inform's development: the system worked perfectly well without. Inform's presently rich support for determiners (see §1b for further bragging) is thus owed to an examination of the interplay of natural language and predicate logic.

But the most visible ingredient in predicate logic is the predicate itself. In

Exists x : *person*(x) and Exists y : *room*(y) and *dark*(y) and *in*(x, y)

("someone is in a dark room"), *room* and *dark* are examples of unary predicates, properties which are either true or false about any given thing, while *in* is a binary predicate, whose truth depends on a pair of things.

I suggest that an examination of the role played by predicates in IF points up failings in today's IF design systems, or to put it another way, opportunities for tomorrow's. I chose *room* and *dark* and *in* for the example above because they are easy to express in almost every IF design system whereas, say, *peacock*, *striped* or *reminiscent-of* are not. While deciding which predicates ought to be built into an IF system is an important question – closely related to the discussion of kinds in §2a – that can easily be a distraction from a subtler but perhaps more important issue: what order do the predicates in an IF system have? Are they unary, binary or higher-order still?[21]

The original Crowther and Woods adventure game program (*c.*1977) essentially only had unary predicates: though there was a kludge to implement the famous bird-in-the-cage puzzle, the game had a limited grasp of spatial relationships. It relied on what we might call a unary predicate *carried*(x) rather than having any general notion of *in*(x, y). As a result only the player could carry things. This failing was remedied from the very first, MIT lab-written implementations of Zork (*c.*1978), bringing in the containment tree we use today, or from our point of view the first binary predicate: *in*(x, y). At some point early in the 1980s history of the commercial IF company Infocom, a second binary predicate *on*(x, y) was added: something could be "on" a table, and as a result respond to different actions from something "in" a box. For reasons I have argued in §2a, it is perhaps

21. Indeed, it seems to me that a good question to ask about any conceptual model of the world is: of what order are the predicates? For instance, one way to describe the historical development of particle physics would be to note that while physicists would ideally like to minimise the number of unary predicates in their descriptions of reality – *electron*(x), *photon*(x), and such – that is nothing like so earnest as their wish to reduce the number of binary predicates: the fields of gravity, electromagnetism and so on.

natural that these binary predicates were chosen first, but there are surely others in the world: yet design systems such as Inform 6 make no particular provision for them (except for what might be called *part-of*(x, y), and that is confusingly implemented). Moreover, binary predicates which ought to exist in Inform 6 do not, because they are wrongly implemented as unary ones. Just as we might regard *carried*(x) as being *in*(x, player), and say that the trouble with the Crowther and Woods program was that it allowed *in*(x, y) only for one privileged object y, so Inform 6's unary predicate *worn*(x) is really an inadequate implementation of what ought to be a binary predicate *worn-by*(x, y). The result is a world model for IF which fails to distinguish between clothing and possessions for everybody except the player-character. More generally, the reduction of a binary predicate *b*(x, y) to a unary one *u*(x) which implicitly takes y to be the player-character – as with *carried* and *worn* – encourages a style of IF heavily centred on the protagonist.

A second limitation is that contemporary systems for IF allow the free creation of new unary predicates (attributes, as Inform 6 calls them: boolean properties, some would say): *valuable*(x), say, or *large*(x), just as we like. Most designers of IF would regard this as an essential tool. But there is little or no built-in support for the creation of new binary predicates. This is no great problem with inanimate objects, because they interdepend so little. It becomes a more serious restriction when dealing with objects which have internal states relating to other objects: in fact, to objects which have what we might call knowledge of the world around them: which is to say, to people. People have internal states very much based on other objects and people in the world. The things they recognise, the places they have been, the people they know. Throughout the first year of the Inform 7 project, I kept feeling that we should add some convenient way to implement people having memories, or knowledge: I now believe that this is a special case of a more general need to be able to create binary predicates.

Inform calls these "relations", and allows us to teach it new verbs which express them. The addition of this ability to Inform marked a dramatic change in the nature of the language: it almost immediately seemed impossible to imagine Inform without relations. For instance, there was a breathless rush of power at being able, for the first time, to write sentences like these in Inform source text:

> Elizabeth loves Mr Darcy. Darcy is suspicious of Mr Wickham.

Underlying these sentences are the binary predicates *loves* and *suspicious-of*, so the ability freely to create binary predicates was crucial.

As a final note in justification of the importance of binary predicates, consider:

> "if a policeman can see a gun which is carried by a criminal..."
> Exists x : *policeman*(x) and Exists y : *gun*(y) and Exists z : *criminal*(z) and *can-see*(x, y) and *carried*(y, z) ?

Note that the two binary predicates are the glue holding this proposition together: with only unary predicates, the variables could not interact, and the language would be much the poorer.

We have seen that binary predicates are sometimes reduced to unary ones, losing some of their expressive power in the process: thus *wears*(x, y) becomes *worn*(x). It also turned out to be interesting to teach Inform to go the other way, and to expand certain unary predicates to binary ones. This is how Inform handles comparatives. For instance, if we

define somebody as "tall" if they are 6 feet or higher, that gives us only a unary predicate *tall*(x): a person is tall, or not. But when Inform reads such a definition, it automatically also creates the comparison "taller" (e.g. taller than 4 foot 6), which is a new binary predicate *taller*(x, y). It also automatically creates the superlative "tallest", which in logical terms is a new quantifier:

"if the tallest person in the ballroom is a man..."
Tallest x such that (*person*(x) and *in*(x, ballroom)) : *man*(x) ?

This is a still further generalization of quantification, and illustrates again that quantifiers also matter.

I have argued that allowing Inform writers to create new binary predicates, not just new unary predicates, made an enormous difference to the expressive power of the system. How about going up from binary to ternary, or even higher-order predicates? Here I think gains are less dramatic. For one thing, binary predicates already permit propositions involving an arbitrary number of variables. For another, most interactions in the real world involve pairs rather than triads. Still a third reason is that storage costs become increasingly burdensome if we want to implement an arbitrary relation between triplets of objects: 100 objects means 1,000,000 bits of data, in the worst case. With that said, it is worth noting that most IF systems do indeed have a ternary predicate: *map-connection*(d, x, y), which asserts that a route exists in direction d between rooms x and y. This is sometimes seen more clearly in its lower-order reductions: *connects*(x, y), say, which really means Exists d : *map-connection*(d, x, y), that is, "there is a map connection between x and y"; or even *adjacent*(x), which means Exists d : *map-connection*(d, x, current location). The full ternary status of *map-connection* is not always recognised, and the failure to store it in a data structure at run-time which reflects this status has sometimes made it quite awkward to create new directions in IF design systems (Inform among them, at present). Inform also has a further ternary predicate *same-property-value-as*(x, p, y), used in the comprehension of descriptions like "people the same age as Henry": here again, though, since p is seldom quantified over, this is really a collection of separate binary predicates, one for each property. A construction like "people who share some property with Mr Darcy", which would require quantification over p, is not currently allowed in Inform.[20]

It might be said that these arguments are all very interesting for Inform, because Inform is based on natural language, but of limited application elsewhere. I increasingly think not, however. I feel that any IF design system, even one based on procedural C-like code or a point-and-click constructor approach, could benefit from a similar study. Indeed, if we were to try to write down a systematic way to evaluate how powerful IF design systems are – for the sort of comparative review which the newsgroups used to be full of, in the early 1990s – the questions we might ask of any given system could usefully include: how can one quantify? how many variables can occur in a conditional question? and what orders of predicate can be created?

§2c. Model theory and discourse representation

In §2b we saw the success of predicate logic as a representation for, especially, questions which can be asked about the current state of the world – questions implicit in instructions such as "if a policeman can see a criminal, ..." then do such-and-such. In this section we consider not questions but the declarative statements which create the world of an IF. This is a different problem: we have much simpler logical propositions to deal with – "Peter is a man.", say, or "Peter is in the treehouse.", each a single predicate with no quantifier – and we are in the present tense, and must be explicit. (Inform does not allow us to give it teasing clues like "Exactly four women are in dark rooms.") On the other hand, we have a great many sentences to cope with, and have to integrate them into a coherent picture of the world. Again, we turn to a mathematical methodology: model theory.

Model theory may be summarised as follows. Suppose we have a collection of statements whose individual meanings we can grasp, but which may or may not affect or even contradict each other. We aim to test the compatibility of the statements, and also to find the simplest meaning which can be given to the whole, by constructing the smallest possible model for them: that is, the smallest configuration we can make in such a way that the text can be seen to be true of it. If no such model could possibly exist, then the text is nonsensical; but if it can, then the text is meaningful. This describes what Inform does quite well: the output is exactly a model of the assertions in the source. For example, given the source text:

> The red box is on the table. In the box is a coin.

Inform converts these to predicate logic, but this does not take us very far:

> *on*(the red box, the table)
> *in*(a coin, the box)

In model-theoretic terms, Inform must now construct a universe set U and an interpretation function v, together with a choice of which predicates *on*(x, y) and *in*(x, y) will hold, in such a way that the two sentences above are both evaluated as truthful for this interpretation. This mathematical description gives us a more precise idea of the principle airily called "Occam's razor" in the Inform documentation: we aim to minimise first the size of U and then the number of pairs (x, y) $\in$ U×U such that *on*(x, y) or *in*(x, y) hold.

Inform expresses its output as a computer program, but if it were put into these mathematical terms (and its internal data structures do indeed have this shape) then the above sentences would produce the following model:

> U = {O1, O2, O3}
> v(the red box) = v(the box) = O1; v(the table) = O2; v(a coin) = O3
> on(O1, O2), in(O3, O1)

(Here O1, O2 and O3 are three different objects.) How is this done? On the face of it, the problem is circular: we need v to make sense of the text, yet it is the text itself which defines v. Moreover, the interpretation function v is in no way an exact correspondence between textual descriptions and members of U: in the above example, two different

descriptions both map to O1. (A valid model does exist where "the red box" and "the box" are taken to refer to different objects, but that fails Occam's razor, because it makes a universe set U which is larger than necessary.) Moreover v cannot be a simple look-up table or dictionary, because in a universe containing both a beech tree and an oak tree, v(tree) would have different meanings at different points in the source text: so it depends on context. In fact, though, the problem of building v and U is fairly easy, by observing clues such as articles and looking for names seen before. Inform works incrementally, first finding a model for sentence 1, then extending it to a model for sentences 1 and 2, and so on. There is no look-ahead to future sentences.

The next stage reduces – or it might be said expands – each sentence into what, in a rudimentary way, is a discourse representation vaguely in the sense of Hans Kamp (1981); though see also Catherine Emmott (1997), whose book analyses genuine excerpts of fiction from a point of view not so different to Inform's.

The essential point is that we can only progress by recognising that a sentence contains both explicit and implicit meanings, so (despite the optimistic talk of §2b above) it is inadequate simply to replace the sentence by its mathematical analogue. From each sentence S, a set of "inferences" is extracted which semantically entail it, that is, such that a human reader will agree that S is a true statement about any model in which the inferences all hold. For instance, Inform reduces "In the box is a coin" to the discourse representation

$$\mathit{in}(O3, O1), \mathit{contains\text{-}things}(O1), \mathrm{Not}(\mathit{room}(O3))$$

That last inference, "it is not true that O3 (the coin) is a room", is typical of the unspoken assumptions in English which Inform stores in discourse representations.

Unfortunately, these are not typical sentences. What are we to do with "A door is always open."? One answer would be to store this as

$$\text{For all } x : \mathit{door}(x) \text{ and } \mathit{open}(x)$$

but, especially in the initial model-building stage of Inform, we want to avoid quantifiers like "for all" if we possibly can. (That way lies reasoning, which is difficult.) We get around this by treating the unary predicates *door* and *open* as being different in character. For instance, the following text:

The portcullis is a door. The portcullis is closed.

is not, as might be expected, represented as $\mathit{door}(O4)$, $\mathrm{Not}(\mathit{open}(O4))$. This is because Inform divides unary predicates into two sorts: being a door is the "kind" of the portcullis, an immutable and basic characterisation, certainly true or certainly false; while being closed is a mere "property", which we regard as less important, as something which might change in play later, and as something subordinated to kinds in any case. (A door can typically be closed but an animal cannot.) Inform therefore constructs a universe set which is a disjoint union of a set of objects O and a set of kinds K, with a kind function k which specifies the kind of something:

$$U = O \cup K, \quad k: U \to K.$$

For instance, if v(a door) = K5 then our examples so far have the model $O = \{O1, O2, O3, O4\}$, $K = \{K5\}$ and the discourse representation of "A door is always open" becomes

simply *open*(K5). This means that the inconsistency of

A door is always open. The portcullis is a door. The portcullis is closed.

can be detected without formal reasoning. By a simple substitution we accumulate the facts about the portcullis as being *open*(O4), Not(*open*(O4)), which is easily picked up as a contradiction. These methods enable us to ensure the continuing consistency of our model world, or else to report the proper errors.

It might be objected that a few propositions are inescapably universal: for instance,

For all x,y : *in*(x, y) implies Not(*on*(x, y))

If we want never to store universals, what shall we do with this? In principle we could handle it by extending our discourse representation for "In the box is a coin" from *in*(O3, O1) to *in*(O3, O1), Not(*on*(O3, O1)) but it is inefficient to keep generating these extra inferences. Instead, Inform knows that certain binary predicates have restrictions attached (the relation corresponding to *in*, for instance, is in Inform parlance a "various-to-one" relation), and Inform checks these restrictions explicitly when it looks through the "knowledge list" about an object in the model. It looks for a few more subtle problems than the blatant contradiction in the example of the portcullis above, too, but all of this is essentially an implementation issue.

A detail omitted above is that Inform records the predicates in a discourse representation with a degree of certainty attached – impossible, unlikely, likely or certain. (There is actually a fifth state, "don't know", but there is no point recording what we don't know in the discourse representation.) Inform uses these certainty levels mostly to handle adverbs like "usually", but also in reading lines like:

East of the Grove is the Temple.

whose discourse representation includes **certain:** *map-connection*(east, Grove, Temple) and **likely:** *map-connection*(west, Temple, Grove). The merely likely nature of the latter predicate means that no contradiction will be generated if we later find that **certain:** *map-connection*(west, Temple, Sanctum), because only a clash of certainties is reported as a contradiction. To my surprise this turned out to be both adequate and easy to implement, adding almost nothing to the complexity of the whole. It turned out that simplistic methods were perfectly good and that no formal system of fuzzy logic was required.

Generalized cardinality quantifiers (see §2b) have been fruitful in model-building, too. They enable declarations such as "Six soldiers are in the barracks", which in logical form read back as

At-least-6 x : *soldier*(x) and *in*(x, barracks)

"At-least-6" looks vague, but Inform applies Occam's razor and makes exactly six. (Similarly, if we define "tall" as meaning a height of at least 6 feet, and then declare that "A tall man is in the barracks", he will be created with a height of exactly 6 feet, the minimum requirement to qualify.) But just as §2 began with a comment on the subtlety of pronouns and their meanings, as an illustration of the difficulties involved in semantics, so it will also end. Inform may have stocked its predicate logic with an exotic range of new quantifiers, but the rules governing those quantifiers are still only the standard ones for predicate calculus. If we have "For all x, such-and-such" as one sentence, then the next

sentence is not able to refer to the same x, because the x is bound to the limited scope of the "For all" which we have left behind. That makes it difficult to accommodate text like "Every man has a weakness. It need only be found out." It may be that the best solution is to relax the rules on the binding of variables by quantifiers, which is essentially Kamp's solution to the problem of pronouns, but I wonder if it isn't better either to (a) disallow the use of pronouns except in limited circumstances (the current solution), or (b) use an ad-hoc system for pronouns which does not try to accommodate them into the predicate calculus. Years of kicking the Inform parser this way and that, in its implementation of pronouns, has made me more sceptical than most philosophers of language seem to be about the "principle of compositionality", that the meaning of a whole discourse is a function of the meanings of its constituent parts. Maybe the mind also handles pronouns with a poorly implemented lookup table, quite separate from its parsing of the rest of sentences: who knows.

To read today's philosophers of language is to become aware that there are very clever people, with enormously lucid gifts for expression, who after millennia of work have almost no idea of how the mind of a writer works. But that does not mean they have nothing to teach the designers of computer software, and in particular software which helps an artist to create a new work. In a very modest way, in a highly simplified "toy" environment, an IF design system is also an attempt to see what are the natural ways in which a writer imagines and expresses a situation: the closer it gets to this goal, the easier it will be to use, and the more powerful its results. If there is one lesson learned which I wish to record about the Inform project, it is that the professionals – philosophers and linguists – do know what they are doing, and are worth listening to. I spent the middle year of the project reading: it is how I should have spent the first.

Conclusion

Whether Inform 7 will be found useful, or whether it will join the zoo of improbable and neglected design systems through the ages which forms the most melancholy part of the IF-archive, others will tell. At time of writing, its user base can be counted on the number of fingers I type with, which I may say is fewer than ten: the four or five most substantial works of IF written in Inform 7 run to about 60,000 words each, so the total quantity of prose passed through it is still no more than might be found in a long novel. But regardless of how Inform 7 is ultimately received, I hope to have demonstrated in this paper that the practical experience of recasting IF into natural language has been highly suggestive of what might emerge as a theory for IF design. Work on semantics can help us by identifying what is important in conceptual pictures of the world, and therefore in stories, and therefore in interactive fiction.

References

ALLEN, JAMES, *Natural Language Understanding* (Benjamin/Cummings, 1995).
BARWISE, JON, and COOPER, ROBIN, 'Generalized quantifiers and natural language', *Linguistics and Philosophy*, 4 (1981), 159–219.
DAVIS, STEVEN, and GILLON, BRENDAN S. (eds.), *Semantics: A Reader* (Oxford University Press, 2004).
CULICOVER, PETER W., *Principles and Parameters: An Introduction to Syntactic Theory* (Oxford University Press, 1997).
— and JACKENDOFF, RAY, *Simpler Syntax* (Oxford University Press, 2005).
EMMOTT, CATHERINE, *Narrative Comprehension: A Discourse Perspective* (Oxford University Press, 1997).
GODDARD, CLIFF, *Semantic Analysis: A Practical Introduction* (Oxford University Press, 1998).
HERTZFELD, ANDY, *Revolution in the Valley* (O'Reilly, 2004).
JACKENDOFF, RAY, *The Fundamentals of Language* (Oxford University Press, 2002).
JESPERSON, OSCAR, *A Modern English Grammar on Historical Principles* (Allen and Unwin, 1909).
JOHNSTONE, P. T., *Notes on Logic and Set Theory* (Cambridge University Press, 1987).
KAMP, HANS, 'A theory of truth and semantic representation' (1981), reprinted in Davis and Gillon (2004).
KERNIGHAN, BRIAN W., and PLAUGER, P. J., *The Elements of Programming Style* (McGraw-Hill, 1974).
— —, *Software Tools* (Addison-Wesley, 1976).
KNUTH, DONALD, *Literate Programming* (Addison-Wesley, 1992).
—, and LEVY, SYLVIO, *The CWEB System of Structured Documentation* (Addison-Wesley, 1994).
KÜBLER-ROSS, ELIZABETH, *On Death and Dying* (1969)
MONTFORT, NICK, *Twisty Little Passages* (MIT Press, 2004).
NELSON, GRAHAM, *The Inform Designer's Manual*, 4th edn. (The Interactive Fiction Library, 2001).
REICHENBACH, HANS, 'The tenses of verbs' (1947), reprinted in Davis and Gillon (2004).
SPOLSKY, JOEL (ed.), *The Best Software Writing I* (Apress, 2005).
STALLMAN, RICHARD, *GNU Coding Standards* (Free Software Foundation, ongoing: October 2004 edition consulted).