Animate Z using the Codd Logic Programmining Language.

This talk is concerned with the practical aspects of PhD theses. The first of two seminars describing work presented in my thesis is the spring 2004.

FACS - AGM, April 23rd 2004

Margaret West

Programmining Language: Method and Examples

Animate a Z Specification in a Logic
lead to loss of human life or significant financial losses. Integrity systems for failure of a system component may ultimately fail. System implications are particularly important for high-overall system performance, all the effort goes into its design. The differ from electro-mechanical systems in that software does not have a production phase; all the effort goes into its design. The Engineering processes for systems which have embedded software
on (say) sets and logic.

An alternative is to write the specification in a formal language based

language is that such a specification can be ambiguous. An

A problem with requirements which are written in a natural

Requirements and Analysis, Design, Coding and Testing.

A typical life-cycle for a computer based system consists of Systems

Software Lifecycle
latter shown to be correct with respect to the design.

Coding: The design is further refined to become code and the
description its functionality and defect laws.

Animation: A version of the specification is executed in order to
reasoning can then be applied to check the consistency of the
Advantages are:
A project, CZT, is currently underway to develop on-line tools.

currently embeeded at a site in Europe.
crossing was developed using B with its tool support and is
B AMN has also been used - for example software for a level

in the UK - by myself and others during the EC funded DRIVE
and validation and to cure problems in Pelican crossing equipment
development process in Europe, USA and elsewhere. It was used to

Z is used by industry as part of the software (and hardware)

than B but B has a much better tool base than Z

are Z (pronounced, and) and B AMN. Z is more flexible
examples of formal notations based on sets and logic (ZF set
Properties are present or absent.

- Animation can demonstrate whether some domain specific
  animation can produce counter examples.
- Animation can produce counter examples be exhaustive.

- Compensates for the fact that tests used for animation can seldom
  Animation is a form of testing, whereas formal proof
  Animation and formal reasoning are used to validate the
  never be said to be correct - it can only have increased quality.

Validation: Animation and Proof: A formal specification can
We can also declare global constants as in the Makefiles below:

```
constants on data
```

```
data declarations
```

```
Some Typed Sets
```

The typical sets of operations on the specification
structure mechanism for $\mathcal{Z}$ is the schema which is based on $\mathcal{Z}$
notation is based on typed set theory and first order logic.

The
<table>
<thead>
<tr>
<th>Field</th>
<th>Count</th>
<th>Count : 0 . MaxFields</th>
</tr>
</thead>
</table>

of Fields and the latter a number. The former is a finite subset of Fields. The count of the Fields is `Count`. We define the field system in terms of its state variables which are.

\[ \text{MaxFields} : \mathbb{N}^1 \]

Example: A small field system involves a single given set:

\[ [\text{Field}] \]
For example - representing the operation of adding a file, and we can declare schemas in the upper part of another schema. Schemas can be thought of a possible set of variable/value bindings.

\[
\begin{array}{c}
\text{Files} \equiv \text{Count} \\
\text{Count} \equiv 0 \cdot \text{MaxFiles} \\
\text{Files} \equiv \text{Field}
\end{array}
\]

... system after some change where all variables are primed. This specification is structured according to the finite state machine model so that by convention Files' represents the machine model that by convention Files' represents the
are expressed in terms of the old ones.

\[ \text{A new file is input and the new values of files and count} \]

\[ \text{Pfiles, } \text{Pfiles', } \text{Pfiles' or } \text{Pfiles} \]

\[ \text{The predicate of ADDPID also includes the predicates of} \]

\[ \begin{array}{c|c}
\text{Count} = \text{Count} + 1 & \{ \text{NEWFILE} \} \cap \text{Pfiles} = \text{Pfiles} \\
\text{NEWFILE} \notin \text{Pfiles} & \text{Count} > \text{Maxfiles}
\end{array} \]

\[ \text{The declarations of } \text{Pfiles', } \text{Pfiles' are included.} \]

\[ \text{NEWFILE} : \text{Pfiled} \]

\[ \text{Pfiles', } \text{Pfiles' } \]
\[ \text{Count} < 3 \Rightarrow \text{Files} \in \{F_1, F_2, F_3\}, \text{Files}^\prime \in \{F_1, F_2, F_3\} \Rightarrow \text{Files} \]

This is expressed as a binding thus:

\[ \text{files} = \{F_1, F_2, F_3\} \Rightarrow \text{files}^\prime = \{F_1, F_2, F_3\} \]

the assignment:

A binding of ADDPID associates values of the schema variables to their names.
See next slide.

I would also add - the ability to replicate the modular nature of Z:

- Correctness
- Efficiency (performance of the animation)
- Sophistication (less likely to go into an infinite loop)
- Coverage (of the Z grammar)

Requirements for an Animation (established by BB94)
Every conjunct in a schema predicate should be exercised.

Lower level schemas, schemas which call other schemas and unit testing of schemas, schemas which call entire test cases. Integration testing involves interfaces between and includes integration testing involving interfaces between.

White box testing involves the structure of the specification.

Initial and terminal state(s).

Values for input, input with safety or integrity implications, any accept data from the containing system (for example) boundary schema which model top level functions and which therefore schemas which model top level functions and which therefore.

Black-box testing is designed to include tests which check software testing into black-box testing and white-box testing.

Tests for animation purposes can be divided (as in the case of Test Strategy for Animation:}
A better choice is the logic programming language Prolog and a method [WHJ95] was implemented using the method [W92] and the Z specification of the Pelican equipment was contained in [W92]. A method of translating from Z to Prolog is contained in [W92].

Because of the ability to express a relation, the logic program has been chosen.

Choice of Language: A logic programming language is often
modules (and can themselves import (other modules).

The language is modular and modules can be exported to

- Flexible computation rule - user-defined control declarations
  •
  - Calls to negative literals are ground.
  •
  - A set data type is supported.
  •
  - The sorts in Gödel become the types of Z.
  •

The advantages are that:

Gödel is strongly

must be sound with respect to this semantics. Gödel is strongly

Gödel: Designed by Hill and Lloyd [HL94].
Predicate Definitions:

A predicate definition consists of a declaration, specifying the type(s) of its arguments, and a set of statements of the form where ≤ in Gödel means "if" and in contrast to Prolog, upper case is used for constants and lower case for variables. Head is an atom with the defining predicate and Body is a formula in first order logic and may be absent. Body can include first order constructs such as universal and existential quantification.
\begin{verbatim}
% etc - all subsets will eventually be output
\%
{1, 2, 4} = x
{1} = x
{} = x

\{1, 3, 4\} \rightarrow x \text{ Subset} \{1, 3, 4\}, 2

x \rightarrow 2
x \rightarrow 1
x \rightarrow \\

\% In \{1, 2, 4\}.$ \text{In means set membership.}$

\% In [Demo1] is the prompt.

\{x : x \rightarrow 1 \rightarrow 5\}
\% Intensional: \{x \rightarrow 1 \rightarrow 5\}
\% Extensional: \{5, 6, 7\} \rightarrow \{6, 7\} \text{ or}
\%
Sets in Codel, sets can be extensional - equality takes no account
\end{verbatim}
\[ \{\{5\},\{2\}\}\{1,2\}\{1,5\}\{5\}\{2,5\}\{1,2,5\}\{\{1\},\{\}\}\}=x \]
\[ n = 10, \]
\[ P \subseteq \{10 > z >= 1 \}
\[ \forall 0 = z \mod n : \{z\} \subseteq \{s\} = x \rightarrow [\text{Demol}] \]

% Queries can be presented as conjuncted predicates:
PredicatE PP : Set(0(p`(a`,p)) * Set(a) * Set(p)).

%% declaration of partial function %%%%

\[
\text{function ordpair : a \times b \rightarrow op(a,p) .}
\]

\text{Constructor op/2.}

a relation between two sets.

the definition of ordpair and hence of constructor, op which allows the definition of ordpair and hence of constructor (in a Gödel module) An important part was a type myself) in a Gödel module Tlp. An important part was a type to be modelled using Gödel. All of these were constructed (by involving two given sets (i.e. relations and functions) which we have already seen. They also include pairs of values, Z data types include set membership and subset of a given set, Library
\[
\forall \lambda [\forall u (x' \in \text{ordpair}(x,u) \land \forall y (\lambda z (x' \in \text{pt} \land z = \text{ordpair}(x',y)) \land \forall w (\lambda x' z (\text{pt}(w, s_1, s_2) \rightarrow \text{ALL}(z, X) \rightarrow \text{ALL}(z, Z))]
\]

% query and answer: partial function is checked

% This determines whether a set is

\[
\forall \lambda [\forall u (x' \in \text{ordpair}(x,u) \land \forall y (\lambda z (x' \in \text{pt} \land z = \text{ordpair}(x',y)) \land \forall w (\lambda x' z (\text{pt}(w, s_1, s_2) \rightarrow \text{ALL}(z, X) \rightarrow \text{ALL}(z, Z))]
\]
USER, SQL

Create, Name, % schema names

Create, Count, NewField, % variable names

Field, Field, Field:

CONSTANT

Given set values, schema variable and schema names

% Given, Name, Var, BindVar.

Field, Name, Var, BindVar.

BASE % Field is specific to the example

Schema and Variable Names

setfield = Field, Field, Field;

{ Field } -> setfield = Field, Field (E).

{ Demo[2] ]

To collect together the values.

Introduced to model the environment. An extra predicate is needed

Given Sets: The given sets of the specification are declared as one

Translation Rules - Fragment
where binding is of type List(Variable) and name is of type Name.

\[
\text{Predicate(binding, variables)} \\
\text{SchemaType(binding, name)} \Rightarrow \text{Signature(binding, variables)}
\]

A schema is modeled by:

**Schema Bindings:**

- For uninstantiated: Bind(F1, F2, f1, f2).
  - Bindvar can be instantiated: Bind(F1, F2, f1).

FUNCTION Bind : Var * Set(Fields) -> Bindvar.

Bind achieves this for Field:

Bindvar, used to facilitate the binding formation of schemas.

**Schema and Variable Bindings:** A further BASE type is
For example, the schema named \textit{Flees}' has schema clause head as \
\begin{verbatim}
  SchemeType [ Bind1(Dest(Flees), Flees), Bind2(Dest(Count), Count) ]
\end{verbatim}
\textit{Flees}'schema and variable names are decorated via Godel functions, thus 

\begin{verbatim}
  Predicate SchemeType: List(Bindvar) * Name.
  SchemaType(Binding, name) is defined by a Godel Predicate:
\end{verbatim}
Card(titles, count).

count In \{ y \geq 10 \} \ &

titles SubSet sEtpID \ &

\( sEtpID = \{ x : \text{IFtitle}(x) \ & \% \% \text{schema predicate} \)

\( \rightarrow \% \% \text{schema binding} \)

\( \text{schemaType}( \text{Bind}(\text{titles, titles}), \text{bind2}(\text{count, count})), [ ] \) \%

\% schema for state titles

specification.

Fragment of the code which models the small file system:

The following provides an

Actualization of the Small File System:
Possible subsets are generated and the predicate checks their values.

In this case the schema declarations

The queries are with respect to the bindings of schemas,

```plaintext
count = count + 1
T(tes) = T(tes) + newT(test
T(tes \in T(tes) \land 
newT(test \in test
\{ x : ISFIELD(x) \}
\land count > 10
% other bindings - committed ..
% schema for operation AddPID
```

Testing subset SetPID and T(tes) subset SetPID mean that all

FileSys, AddPID.
% second schema binding
\[ p \text{[} \text{Bind}(\text{Dest}(\text{count}, 1), \text{Bind}(\text{count}, 0), \text{Bind}(\text{count}, 0)) \text{]} = p \text{[} \text{Bind}(\text{count}, 0), \text{Bind}(\text{count}, 0), \text{Bind}(\text{count}, 0)) \text{]} \]
% first schema binding
\[ p \text{[} \text{Bind}(\text{Dest}(\text{count}, 1), \text{Bind}(\text{count}, 0)) \text{]} = p \text{[} \text{Bind}(\text{count}, 0), \text{Bind}(\text{count}, 0), \text{Bind}(\text{count}, 0)) \text{]} \]

% the initial state
\[ \text{[} \text{Dest} \text{]} \rightarrow \text{SchemaType}(p, \text{Fittes}) \]
% test of schema Fittes

% and the possible inputs which are associated.
% all states will be generated eventually
Hayes [Hay93], and are more complex. The next two examples are case studies from required. The above represents an extreme amount of data is generated with eventually.

\[ p = \text{bind1}([\text{fits}, \text{fits}], \text{bind2}([\text{fits}, \text{fits}]), \text{bind}([\text{fits}, \text{fits}], \text{fits})] \]
is never changed.

The following expresses the property that the \( \text{fid} \) of a channel consists of a file identifier, \( \text{fid} \) and a position within a file,

\[
\begin{align*}
\text{CHAN} & : n \quad \text{pos} \\
\text{CHAN} & : \text{fid} \\
\text{CHAN} & : \text{fid}
\end{align*}
\]

position in the file.

A channel is defined which remembers a file and the current position in the file.

In order to support random access to files for reading and writing,

\[
\text{[\text{fid}, \text{CID}]}
\]

also \[\text{Wes95}\].

See also called \( \text{FID} \). Further given sets are channel identifiers \( \text{CID} \).

Further retrieved using the identifiers; the set of all identifiers is

and retrieved using the identifiers; the set of all identifiers is

UNIX File System The file storage system allows files to be stored
A channel storage system allows channels to be stored and

\[ py = py \]

\[ \text{CHAN, CHAN'} \]

\[ \text{CHAN} \]
binding $\mathcal{C}HAN$. 

$\{ N \mathcal{C}HAN \theta \leftarrow \mathcal{C}i \} \oplus \text{store} = \text{costore}$

$0 = \text{posn}$

$c_i \notin \text{dom store}$

$\mathcal{CID}$ : $c_i$

$\mathcal{C}HAN$ \[\mathcal{CS} \downarrow\]

$\mathcal{OpenCS}$

which is output:

The schema $\mathcal{OpenCS}$ denotes the opening of a new channel, $c_i$.
In the following schema, a channel \( \text{cid} \) is closed; the channel must be updated by the removal of \( \text{cid} \), the input channel, have been previously open (\( \text{cid} \in \text{dom} \text{store} \)) and \( \text{store} \) is

\[
\text{store} \ni \{ \text{cid} \} = \text{store} \\
\text{cid} \in \text{dom} \text{store} \\
\text{CID} : \text{CID} \\
\text{CLOSE} \\
\text{CLOSE}
\]
\( f \) in \( \text{tid} \)
\[ \text{posn in position} \]
\[ \text{posn} \ll 0 \]
\[ \text{posn} \gg \{ \text{00} > x \geq 0 : x \} = \text{position} \]
\[ \text{posn} \gg \{ \text{ISPRTID(x)} : x = \text{tid} \}
\]
% Types of variables, position is a natural number

% [ bind1, bind2 ] in bind = \[ bind1 (\text{pid}, f), \text{bind2} (\text{posn}, \text{posn}) \]
% schema \% schema \% schema \% schema

% The code for schemas \CHAV\', open\CS', close\CS is presented.
% variables that are used in the code
% schema predicates represent constraints for
% inc represents the value of variable named
% closes

% variables t, pos, etc.
% schema predicates represent constraints for

...% inc represents the value of variable named
% closes

% declared schemas Chan, Chan,
% declares Open, Close, and
% Open, Close, and

% schemas Open, Close, and
% are partially shown:
The first query investigates the possible bindings for CS which is provided with as binding equivalent to complex data type than file, file.

However the variables are functions, and therefore of a more in that `after` states are explicitly related to `before` states.

The Unix file system is an example of a constructive specification.

Example of Queries to Unix Files
{\text{cs} = \text{OrIdPart}(\text{Ctd1}, \text{Bind}(\text{ fileId}, \text{Bind2}(\text{posn}, 2)))}

\text{bind4(Dest(Storage), \text{cs}[1], \text{OpenC})}.

\text{SchemaPart}\{\text{Id1, Id2, bind4(Storage, cs)}\}

\text{cs = OrIdPart}(\text{Ctd1}, \text{Bind}(\text{ fileId}, \text{Bind2}(\text{posn}, 2)))

\text{unixFiles} \rightarrow \text{[unixFiles]}

one binding.

the full remainders at 0. In this case the answer set contains more than
untouched) ones and the file identifier can be any. The position in
new channel is added and can be any of the remaining
the new query provides the new value of CS
and position 2, and the query provides the new value of CS
further channel. As before, the existing channel has the identifier
query shows the effect on the channel store, of the opening of a
there are no other values which satisfy the predicate. A further
by the new channel, any. The position in the file is 0.
The new channel can be any except C[7] and the file remained.

\[
\begin{align*}
\{ \langle 0 \leq us\text{op}I, \langle pI \leq py \rangle \leftarrow C[7],
\langle z \leq us\text{op}I, \langle pI \leq py \rangle \leftarrow C[1] \} = \text{store}, \\
\end{align*}
\]

Thus amongst many possible values we have:

\[
\begin{align*}
\text{ordpair}(C[7], \text{bind}(\text{pid}, \text{fid}, \text{posn}, 0))
\end{align*}
\]

Initiating a back track gives a further answer:

\[
\begin{align*}
\text{ordpair}(C[7], \text{bind}(\text{pid}, \text{fid}, \text{posn}, 0))
\end{align*}
\]
The next query investigates which channels can be closed:
This was originally animated in Plog in [WEG92].

Assembler:

Equivalent:

means of the following sample of assembly code and its machine
address or location in the machine’s memory. It is analyzed by
integers (opcode and operand) are located at a single machine
The computer is regarded as a “one address machine”, in that two

<table>
<thead>
<tr>
<th>Location</th>
<th>OPCODE</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>0010</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>0001</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>0000</td>
<td>11</td>
<td>8</td>
</tr>
<tr>
<td>0000</td>
<td>12</td>
<td>7</td>
</tr>
<tr>
<td>0000</td>
<td>16</td>
<td>2</td>
</tr>
<tr>
<td>0000</td>
<td>02</td>
<td>5</td>
</tr>
<tr>
<td>0000</td>
<td>03</td>
<td>4</td>
</tr>
<tr>
<td>0000</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>0000</td>
<td>01</td>
<td>2</td>
</tr>
</tbody>
</table>

Exit  Return
JUMP  LOOP
Jump  Exit
Compare 1
Store
Sub
Loop  Load
\[ W, V, W, V, P, O \]

\[ W, V, W, V, P, O \]

and of permissible labels and opcodes: \( W, V, W, V, P, O \) and \( W, V, W, V, P, O \).

The "given" sets are of assembly and machine instructions: \( A \) and \( Z \).

### Assembler and Machine Requirements in \( Z \)

Value in the corresponding machine operand.

An assembly instruction, once directly translated to their numeric

The operand mnemonics, when they appear in the operand field of

The operand field of some instruction, must appear in the label field of some instruction.

Determines the instruction, and conversely every referenced label

If an assembly instruction is labelled, then that label uniquely

Operand is treated as a value or as an address.

A machine instruction's opcode determines whether its integer

Opcode and operand.

An assembly instruction has a maximum of three fields: Label,
equals \( A \).

The domains of ref and num are disjoint (A\(3\)), and since assembly

\[ \text{lab, op, ref, num:} \]

labels, assembly contexts involve the presence or not of symbolic labels.

The first of these machine language requirements and the mnemonics for translation

THREE axiomatic descriptions model assembly requirements.
operand is referential (etc.).

operand of machine instructions when the corresponding assembler
function.

S1: The inverse of 

S2: A label must be unique to a position, the composite

respectively.

output of machine instructions are sequences 

Assembly Process in Input of Assembly Instructions and

\[
\begin{align*}
\forall 6 & \quad \mathcal{V} = \text{dom ref} \cup \text{unm dom} \\
\forall 5 & \quad \emptyset = \text{unm dom ref} \cup \text{dom ref} \\
\forall 4 & \quad \mathbb{N} \leftrightarrow \mathcal{V} : \text{unm} \\
\forall 3 & \quad \mathcal{W} \leftrightarrow \mathcal{V} : \text{ref} \\
\forall 2 & \quad \mathcal{W} \leftrightarrow \mathcal{V} : \text{do} \\
\forall 1 & \quad \mathcal{W} \leftrightarrow \mathcal{V} : \text{lab}
\end{align*}
\]
proves equivalent to the above.

The assembler can also be modelled in two phases - which Hayes

Implementation of the Assembler:

\[
\begin{align*}
S_5' (\mu \nu \theta \phi) \cap (\eta \zeta \xi) &= \text{domain mem} \\
S_6' \text{sem mem} &= \text{do code} \text{sem code mem} \\
S_4 \text{ran sem mem} &= \text{do mem} \text{ran sem mem} \\
S_3 \text{ran sem mem} &= \text{do mem sem mem} \\
S_I \text{ran sem mem} &= \text{do mem sem mem} \\
S_2 \text{ran sem mem} &= \text{do mem sem mem} \\
\end{align*}
\]
their final values in core.

During Phase I, \( t \) will be constructed and the opposite and

\[
\begin{align*}
\text{core : seq} & \quad W \\
W & \leftarrow N : t \\
N & \leftrightarrow \text{WAS} : s I
\end{align*}
\]

We can also demonstrate the equivalence: The first phase is

about this intermediate state is modeled by \( IS \).

which are partly constructed during phase one. The information
of machine instructions captures the state of these instructions,
and \( s I \) which records the values of the symbols. The sequence \( core \)
built: \( t \) which records the positions where symbols are referenced,
constructed for inputs with numeric operands. Two tables are
captured by \( Phase I \), where the machine instructions are

Instructions is completed. This phase the construction of the operands for the output machine.

During operand helds is obtained only from the reference table. During accessible and so information about the symbolic and numeric

During Phase 2 the input sequence of assembler code is not

\[
\begin{align*}
\text{Phase 1} & \quad \exists \text{dom mem} \supseteq (\text{do } \text{save}) \\
\text{Phase 4} & \quad (\text{mem } \text{do } \text{save}) = (\text{core } \text{operand}) \\
\text{Phase 3} & \quad (\text{mem } \text{save}) = (\text{core } \text{operand}) \supset (\text{mem } \text{ref}) \\
\text{Phase 2} & \quad (\text{ref } \text{save}) = (\text{mem } \text{ref}) \\
\text{Phase 1} & \quad (\text{lab } \text{do } \text{save}) = (\text{lab } \text{do } \text{save}) \\
\end{align*}
\]

For the output machine sequence:

The sequence core can be thought of as acting as a place-holder.
is the same as Assembly.

In Hayes’s Implementation is expanded out and it is proved that it

\[ \text{Implementation} \equiv \text{Phase}_2 \setminus \text{Phase}_1 \cap \text{Phase}_2 \text{ (st, rt, core)} \]

IS is hidden for the intermediate stage is not normally significant:

To obtain the final Implementation, the schemes are conflated and

\[
\begin{align*}
\text{P2.4, P2.5} & \quad (t, s) \in \text{sem}_i \circ \text{operand} \quad \supset \quad (t, m) \circ \text{domain} \\
\text{P2.3} & \quad (\text{P2.2} \circ \text{operand}) \supset \quad (t, m) \circ \text{domain} \\
\text{P2.2} & \quad \text{ran rt} \subseteq \text{domain st} \\
\text{P2.1} & \quad \text{N} \leftrightarrow \text{WALS} \subseteq \text{st} \\
\text{sem}_i : \text{sem}_i & \quad \text{IS}
\end{align*}
\]
Scheme to Assembly (as before).

The natural numbers, \( \mathbb{N} \) are modeled by a subset \([0, \ldots, 5000]\). The

\[
\text{Sega, Segm, Lab, Op, Ret, Num, Operand, Mem, Var;}
\]

% variable names

\[
\text{V1, V2, Loop, Exit, Label : Sym;}
\]

% symbols

\[
\text{CONSTANT}
\]

\[
\]

% given sets

% given sets

For example, sets, symbols and variables are as follows:

The predicate definitions are presented in the module \text{Assembly} where the 
signature of \text{Assembly} (as for the small file system) is set.

The data and as above. They are then treated as if they were declared in the

definitions. \text{Assembly Context} \ldots are declared as schemas, named

\text{Translation of Assembly to Code} \ldots the three axiomatic
\[ \text{SchemaType}([\text{Bind}(\text{IN}(\text{SegA}, \text{SegA}), \text{SegB})], \ldots, \ldots, \ldots, \ldots) \]
\[ \ldots \text{op} = \text{Operand}(A3, \text{Load}) \}
\[ \ldots \text{Operand}(A9, \text{Return}) \}
\[ \text{sema} = \text{Operand}(T, \text{M1}) \}
\[ \text{sema} = \text{Operand}(T, \text{A1}) \}
\[ \ldots \text{Operand}(9, \text{A9}) \}
\[ \text{Assembly} \rightarrow \text{sema} = \text{Assembly} \]

Values are calculated.

output stream correctly models the input stream, no new
output assembl output stream is present values for checking that the
The best we can do here is present values for checking that the

A truncated form of a query to Assembly is presented here. It is

\begin{align*}
\text{SchemaType} & \rightarrow \text{Bind}(\text{IN}(\text{SegA}, \text{SegA}), \ldots, \ldots, \ldots, \ldots) \\
\text{Bind} & \rightarrow \text{Operand}(A3, \text{Load}) \\
\text{Operand} & \rightarrow \text{Operand}(A9, \text{Return}) \\
\text{sema} & \rightarrow \text{Operand}(T, \text{M1}) \\
\text{sema} & \rightarrow \text{Operand}(T, \text{A1}) \\
\ldots & \rightarrow \text{Operand}(9, \text{A9}) \\
\text{Assembly} & \rightarrow \text{sema} = \text{Assembly} \\
\end{align*}
Phase I:

also consistent with Phase 1. if and st are also computed during
numeric fields represented after the first phase, the final value is
each value was successful, so although operand need only have
- The final value of operand was also input as part of another query

\{3 \leftarrow W, 100 \leftarrow M1, \text{Operand} \}

where only numeric fields are represented:

as it is implicit), The intermediate value of operand is also input,
and also core (because its definition cannot be computed using

For the first phase, Phase 1, the assembly instructions are Input,

compiled to form Implementation.

Providing the input to Phase 2, and then together when they are
aligned by arithmetic them separately with the output of Phase 1.

The two Phase design of the assembler can also be
fashion. The two Phase design of the assembler are modeled in a similar

Two Phase Design
Yes

% answer is checked and found to be correct.

% [ ... ] [ ... ]

\{ rt = \{ ordpair(3',7'), ordpair(6',8'), ordpair(6',7') \},

\{ st = \{ ordpair(3',8'), ordpair(6',9'), ordpair(6',8'), ordpair(3',9') \} \}

\{ \}

\}

-> [assembly]

... Assembly

... together with values as for Assembly.

For Phase 2, the values of \$t\$ \$s\$ computed by Phase 1 were input,

\{ \}

\}

\}

\}

... Assembly
operand succeeded.

With the addition of core, in this case only the final value of the implementation, the values provided were as for assembly:

```
... -> [ ] Implementation
...  
```

The head only of implementation is shown: and the values of the first were hidden, but core was necessary for the Next, phase 1 and phase 2 were combined to form implementation,
Huddersfield.

rules is shown. The next talk will be at the University of
story continues next month, where the correctness of the
Only the rules are shown - a proper interface would be needed.
Computation is speedy; response to a query is instantaneous.
notation, including its modular nature.
the case studies cover some important features of the
animation and is superior.
The General animation can be compared with the Prolog

Conclusions
References