GLOSSARY

Software Languages

---

action statements types of

Data access (ary's, fields, files & URL's)

Event generation

Test

Transformation: Data conversion, File conversion, Affine Graphics Transforms, Color Transforms, State Machine Transforms, String transforms

Bridge & Functions: using API's of frameworks, commonly expected API's, event src connection

activity networks intermediate language representation in compilers

Compiles often use a form of activity on vertex networks when converting a program into low level instructions. The `activity' is called a basic block. A basic block is an "instruction sequence with no branches into or out of the middle;" branches to the block go to the top, and the bottom of the block has one or more branches (determined by some condition) that connect to the next block(s). The edges may be weighted based upon factors such as frequency of activation, or importance of the branch.

The goal of the compiler is to flatten out this graph into an efficient string of instructions. If the graph was left `as-is' the system would suffer significant penalties from the pointer chasing involved.

algorithms generic

How can you represent efficient algorithms independently of my particular data-representation scheme?

How can you provide an interface to a diverse set of data representation strategies that gives us the flexibility to choose appropriate representations but also allows algorithms that work without detailed knowledge of internals.

Generic algorithms used the interface rather interact with data directly.

API

Application Programming Interface. see *interface*

associativity

See operator

basic block

A sequence of instructions to branch point.

bill of materials problem

Bill of materials

Product mix

Quantity on hand (stock)

Quantity to buy from suppliers. Suppliers offer different price points. Suppliers differ in min/max quantities, identifying suppliers, services agreements.

Query:

```
Select ItemId, Qty
  from B BOM, P Projects
where B . projectID == P . projected

Update from P0, P1
Qty = (select Qty
        from BOM B
```

```
where B . projectId = P1 . projected && B . itemId = P1 . itemId)
```

Backus-Naur Form A notation used to describe the admissible calling sequences for an interface. Traditionally this form is used to define the syntax of a language.

see also *interface*

Backs-Naur Form	Used to describe acceptable calling sequences for an interface
category	A module that implements some calls, but is not guaranteed to provide an interface.
interface	A set of calls that work together, and must be implemented by `objects' in order for the object to say it implements the interface
protocol	

Table 1: *Distinction between Backus-Naur Form, interface, and protocol*

BDD
binary decision diagram A data structure specialized to represent Boolean-logic and software branches. It is common with formal verification of digital designs and some kinds of software. A given path through the system can be represented as a binary string, since each branch is binary. Some analysis employ regular expressions of paths that must be present – or must not be present.

see also bounded model checking, regular expression, satisfaction (Boolean)

behavior allocation "The act of choosing which subroutines to associate with which classes, and which subroutines call which other subroutines." "has the biggest impact on the system."

binding The sense of a symbol having an assigned value.

bound Symbol has a precise value

free Symbol is without value.

BNF see Backus-Naur Form

Boom hierarchy In a hierarchy of types, each level has a definition of three operations:

FILTER: Removes elements from a structure. Each x in S such that P(x).

MAP: Applies a function to all elements in a structure. Each f(x) for all x in S.

REDUCE: Combines elements in a structure.

bounded model checking A formal analysis usually by treating a system's behaviour as state transitions and analyzing until a depth-limited has been reached. The limit is expanded until a rule violation is found. The model is often described as a Kripke structure, treating the system as a set of state machines, which communicate via variables. The state of the system is the state of all the state machines and the variables. The systems state transition diagram can be examined; often it is converted into a binary-decision diagram. Using a combination of many different state models, each simple in itself, may make the problem more practical.

see also BDD (binary decision diagram), Büchi automaton, temporal logic (linear), regular expression, satisfaction (Boolean)

Büchi automaton A state machine, similar to a finite state machine, which can take infinite inputs; it includes a set of initial states and a set of good states. (They are also like Kripke structures, but interpreted slightly differently). They are used in model (or protocol) checking to say that some action will eventually be taken after event y (to find cases where this doesn't happen), or that action x will never occur after the event.

See also bounded model checking, Kripke structure, model checking, temporal logic (linear)

call signature A descriptor specifying the types of each passed parameter.

category An Objective-C term that allows methods for an object to implemented in different files without generating compilation errors (due to missing methods). This allows methods to

be `added' to existing classes, expanding their interface.

class cluster Related classes that are typically grouped by function.

compile To prepare a machine language program from a program written in a calculus (programming language), making use of the overall structure of the program or generating more than one machine instruction for each symbolic sentence, or both as well as performing the function of an assembler.

compiler Gathers information from source files and puts together a program that will work with a given architecture.

Simple compilers translate with less complex optimizations and allocations. Very simple library and framework call injection. More complex compilers include complex optimization and subroutine formation to work with library call substitutions and framework.

see also reduction procedure

data dependency The level of data dependency governs:

Whether or not we can parallelize the algorithm

Whether we can use other techniques to simplify recursion time & space

dynamic Selected code blocks are compiled into native machine code and linked into the program. Selection occurs by automatically frequently used VM code.

incremental A subroutine is compiled as soon or late as possible. The object code is placed into a repository. Errors are stored and retrieved upon request.

complexity The length of the smallest program to reproduce the behaviour.
Chaitlin, Kolmogorov

See also *purity*

measures Complexity measures include length of function, depth of nesting, number of variables in expression.

McCabe cyclomatic complexity CC=E-(N+P)

Complexity = #Edges – (#Nodes + #Connected Components)

Estimate - #Decision nodes + 1

A = #Attributes (fields) in class

mA = # of (methods in class | procedures) accessing given attribute A

m # of (procedures | methods in class)

sum over all the attributes / fields:

$$LCOM2 = 1 - \frac{1}{M * a}\sum mA$$

$$LCOM3 = \frac{1}{m-1}\left[m - \frac{1}{a}\sum mA\right]$$

>1 -> lack of cohesion; danger

see also *purity*

Table 2: *Complexity*

Complexity #	*Riskiness*
1-10	Simple, low risk
11-20	Moderately complex; moderate risk
21-50	Complex, high risk
>50	Untestable, very high risk

compositionality The meaning of a structure, phrase or formula is a function of the meaning of the elements, their mode of combination, and the context. Decomposition is often concerned with finding such a structuring.

context

data normalization rules of

Attempts to reduce redundancy in a database, and provide useful constraints by striving to define the data as a set of relations in which all of the attributes are functionally dependent on only the primary key. Each step of normalizing produces more tables than in higher-precision forms, with fewer columns per table. The rules are:

Eliminate repeating groups

Eliminate redundant data

Eliminate columns not dependent on key

Isolate independent multiple relationships

Isolate semantically related multiple relationships

Note: structuring a model purely based on the normalization rules will yield a poor model. The complexity of the work rapidly rises with the increase of tables, usually by an order of magnitude. Ie, normalizing table A (lets say O(n) access time) splitting it into two tables, would produce an access time of $O(n^2)$.

delegate A link to another object. A delegate is used to modify the behaviour of a primary object. A common use of delegates is as a data source – a means to get the data to display.

See also *notification*.

delegate	Delegates that are implemented as callbacks, are never queued and are faster than notifications. Like notifications, they can be used to transfer important information about the change in state of a source. Unlike a notification, the callback allows return codes from the delegate. Delegates are used to affect the behaviour of the source object.
event	The abstract change in state, or activity. These are often represented as notifications.
message	Messages are the carrier of notifications and event information.
notification	These are the codified form an event. Unlike delegates, they are often queued for later delivery, and multiple objects may register to receive them. Like delegates, they are used to observe and respond to changes in the originating object. Also, like delegates, many systems implement the registration in a publish-subscribe model, with a callback to the receiver to deliver the notification.

Table 3: *Distinction between delegates, events, and notifications*

distributed linker Similar to a conventional linker, it is concerned with connecting software elements in some form of distributed environment, connected via network protocols or buses.

See also discovery, lookup service

Entscheidungs-problem Literally *decision problem* – a question of whether or not one can decide if a statement is true. The Halting problem, where a machine changes what it does to invalidate your prediction (decision), is the classic proof. Most problems are not so perverse, but those that are can be prevented by not allowing such decisions to be known to the system.

see also Halting problem

equation See also expression, function, interpretation, model, propositions, sentences, situations

equation	A relation between two expression
expression	Doesn't have a relational comparison
function	Assigns unique value to each input

Table 4: *distinction between expression*

interpretation	Assigns intention
model	Assigns extension
statement	?

evaluation procedure

A stepwise, often mechanical, process of inferring the values of variables, or it a given statement is true.

see also intermediary language reduction procedure, valuation function

exception

Destructors get called as it traverses back toward a handler. Checked exceptions.

expression

See equation

extensible language

"A base language which provides a complete but minimal set of primitive. Facilities, such as elementary data types, and simple operations and control constants.

"Extension mechanisms which allow the definition of new language features in terms of the base language primitives.

Semantic extensions: introduce new kinds of objects, data types

Syntactic extensions. New notations for existing or user defined mechanisms.

event

The abstract change in state or activity. Used to observe and respond to commands, changes in state, or other activity.

see *delegate* for the distinction from other items.

see *notification* for the embodiment of events.

event handler

Returns false if further processing may be needed; true if the event was completely handled.

Event handler may generate new event putting them into queue.

Event handling's queue discipline. The queue manager needs to support a concept of priority. If there is a mouse click or button press event, it should be processed ahead of (not yet processed) events, like window move, paint, and z-order change. (Why: because the event's where sent relative to what the window is on the screen, but changing the screen and then processing the events is bad.)

formal system characterizing precisely

Syntax streamlining

An arithmetization method (e.g. Gödel numbering)

A definite method of going back and forth between the arithmetic number coding and conventional notation

To make tractable assign id # to the objects of your attention: each symbol, string, well-formed formula, finite chain of those, proof, etc. get such a number

forward progress

framework

The Framework of a language – or a special core section of the language – is of critical importance. The operations in C are translated either into machine code or into calls to this Framework.

The framework is seldom a separate thing from the language – English is not just a mangle blob of German syntax, with each speaker possessing a unique dictionary. While each speaker, each bit of literature, seems to have its own peculiar lexicon, no massive dictionaries are shipped with Ulysses, and there are no standardization committees for the orthography, usage, interpretation and import of a lexicon. This is done for C; without the bureaucracy, utterances within the C language were always creoles interpretable only by the smallest of tribes.

When a C compiler sees a construct such as

```
char A[50]=...;
```

char B[50]=A;

it will automatically include a memcpy() or equivalent. Similar is true for many math constructs, such as division (mapping to _ldiv()). And when the compiler sees

char A[50]={0};

It will substitute in a memset() or bzero()

This mapping is fine and good, except that the rules for this mapping are not well documented. The compiler is dependent upon an interface to the framework, but it doesn't tell the programmer what that interface will be. I personally have encountered this in two ways. First, there may be case use of _ldiv() or is undesirable, for performance, safety or other reasons. Second, when targeting other architectures, it can be a surprise to suddenly discover a new requirement.

function

signature

Signatures of functions to help identify the same or similar:

1. Signature of structure – e.g. constant values are ignored
2. Signature of function – e.g. constant values are used

f1(x) {return x <= 1 ? 1:f(x-1)*x}

f2(x) {return x<2?2:f(x-1)*x}

have the same structure signature but different function signatures.

generalization

An accurate statement in precise language of what was found with respect to the tendencies, relationships, regularities or patterning among variables under study.

Heckel's algorithm

Finds the longest recurring substring.

See also Bentley-McIlroy matching

idempotent

Many calls to the same procedure have the same result as a single call

ILOG

Tools: solver, scheduler, dispatcher, configuration

Terms: powerful, advanced, versatile, easy, clear

Table 5: ***Planning horizon***

Application	Long-term	Published schedule	Operational Schedule
Scope	Strategic	Tactical	Operational
Timesteps	Month	<-Week->Days->	Hour
Drivers	Money	…	Feasibility
Technique	LP	MIP/Hybrid	CP

immutable

The object, file, or resource cannot be modified; hard-linking it is preferred to an actual copy.

see hard-link, mutable

intermediary language

There is some debate whether a model's interpretation should be described in terms of a machine (i.e., an evaluation procedure), or translated into a set of declarative statements that must be used by another model to infer the values. With a single model, or small number of models, it is simpler to use a direct evaluation procedure. With a large number of models, it may be easier to translate each into a more sophisticated intermediary language. This also reduces the combinatorial complexity of translating from one language to another: either you need two translators for every language (two and from the Intermediary language) or $2n^2$ translators.

see also compiler, evaluation procedure, valuation function.

interface

An interface is a defined method of accessing functionality. An object may support several interfaces.

see also Backus-Naur Form

interpretation

interpretation function — Interprets sentences in the language. The language can be very simple or complex. I'm not familiar with any past a Chomsky level 2. The language can be a non-trivial language of any form that can be systematically interpreted.

see also evaluation procedure, intermediary language, valuation function

interval — A range of numbers; has the advantage of a compact representation and a fast set of operations.

iterator — A construct that allows the examination of each item in an aggregate.

Kripke structure — Checks that temporal logic formulae are valid. A counter example is a trace of the system that violates the property. State transition structure; each state is a value at time. All behaviours of the Kripke structure satisfy or violate formula.

LISP — "a compiled Lisp program is no longer Lisp at all. It breaks the fundamental rule of 'formal equivalence of code and data'.." *http://software-lab.de/radical.pdf*

Scheme — There are two levels to syntax in Lisp family. The 'surface' syntax is the parenthesized prefix expressions. The 'deep' syntax is the one recognized by the interpreter (or compiler), which uses an expression's structure to determine how to evaluate it. With macros, Scheme has a transformational grammar, not a context-free grammar.

model$_1$ — A binding of variables to values. See also *satisfaction*

model finder — A satisfaction procedure, finds the bindings of variables (the model) that make the specification true; often a constraint solver (compile and hand to a SAT solver)

modeling language — Expresses structure constraints and behaviour

model$_2$ — A formal framework for using a few central relationships to represent the basic features of a complex system; models discard important elements and philosophical considerations: they are *not* truth. Models are often described by their role, elements, and test of specification error. *Models are "clipped and pruned till they resemble the conventional birds and animals of decorative art." Alfred Marshall.*

Models should be open about the underlying theoretical principles. These principles must have a concrete form in definite algebraic terms. The model should be transparent about its connections, mechanisms, and flow, coupling effects to outputs. It should be easy to tinker with, yet the user should not have to understand exactly how it works. What are the (hereto fore) unseen expectations?

see endogenous variables, functional explanation, Markov model, Poisson model

Table 6: *Distinction between related terms*

Term	*Distinction*
emulation	Imitates the behaviour of a system, without concern for internal processes
evaluation	To assign a value to an expression
execution	A sequence of instruction passed to an external interpreter
interpretation	Assigns interpretation.
model	Assigns extensions – the values and sets
paravirtualization	Similar to virtualization, except it presents the illusion of a device slightly different from the underlying hardware.
simulation	Mimics the behaviour of a system, with a high degree of fidelity to internal processes, state, etc.
virtualization	Effects the illusion of each user of a device being the only user; the multiplexing software typically saves and restores the state context for each user.

Type	*Distinction*
analogical models	
behavioral	Imitates the behaviour of a system, with-out concern for internal processes
declarative models	can represent important aspects of static systems, but dynamic systems are largely beyond their ability. Most tense analysis in modal systems treat histories as points in time with different sets of facts, ignoring change.
idealized models	
Measurement models	Maps measurements to their theoretical constructs
Parametric models	Predicts values, especially when observables and/or actions are primarily numerical.
Phenomological models	
Statistical models	A type of behavioural model based on probabilities
Structural models	Maps causal and correlative links between theoretical variables. Specifies components and interconnection, often a structural model is a specific implementation.

Table 7: *Distinction between types of models*

behavioural model

Describes the system primarily using

Its actions and actions of its components,

Its interaction with the outside world,

Interactions of its components,

Causality relation

Describes the function and timing, independent of a specific implementation.

see also functional explanation

economic models

Modeling economics poses a challenge since economic relations are very vague. Relationships only have a topology, but no definitive structure. (Does a rise in output, mean a small linear change, exponential, or a probability?) This means the integration of changes will be way off. The relationships may be wrong, or purely ideological; they may be correlative for a while, but the correlations may disappear once the state or other factor tries to manipulate them. Can't predict results based on the results under an old regime.

Many of the elements are linked in a complex system of symbolic equations. They are not sufficiently independent or isolated to examine a subsystem; to solve one part, you need to solve all of the equations simultaneously. Easy to have results that cannot be predicted with naïve models. The messy transitions of the real world are not predicted.

There are genres of economic models. Macro-economic models to demonstrate the circular flow of the economy. Computable General Equilibrium models: these focus on the underlying structure of the economy, ignoring business cycles variations. They can capture one-off difference policy but not the recurring, continuing effects.

identification

Constructing a model by parts and specification

limits of models

Models are not independent checks of their creators: models largely exist to codify a view. Some limits include: experts have their own incentives, there is a high demand for models, no matter their quality. Model selection and designed is to confirm the researcher's ideology, based on (in part) topological and structural changes.

models

A model represents a particular context in which a little algebra is evaluated – a

in logic	system of axioms, operators, rules for combining variables and operators into formulae, a set of entities, their properties and relationships, and a specification of the language relates to those entities and relationships, constraints on what properties there are. These models allow only deductive logic. see also valuation function
non-standard	Alternative interpretations. Try to rule out those interpretations with ambiguity, although this can be hard to spot. Things other than intended may be well described by the model.
numerical model	Numerical models provide numerical answers to policy questions.
partial model	Only can evaluation some statements.
physics	Series of equations of state, relationships between material bodies, and describe their movement, action, behaviour, etc. This is usually divided into parameters, expressions, functions, geometry, coordinate system, materials, analysis.
satisfaction models	In order of increasing difficult: parameters are independent; pairwise; all pairs.
structured models	Means of evaluating a model's quality and characteristics.
model theory	Concerned with making models of a theory. A theory has a model if and only if the theory is consistent. Such a model is a language with an abstract algebra to implement the semantics. An interpretation function that maps language elements to constants, functions, and predicates. The description of the language is often a table with the syntax and how to evaluate predicate phrases of that syntax. The syntax: the kinds of variable (if the language is typed) and how they combine with operators and other variables. The set of entities allowed may be more than a variable – it may include more complex noun phrases, e.g. GlobalCheckFor $var. Discussions of such models focus largely on the syntax (esp. *well-formed formula*) although the issues with interpreting meaning and finding satisfactory solutions is of greater importance in the long term (a language is learned 'once' but used for a long time), and more difficult. see archetypical language understanding, evaluation
model world	Composed of A set of possible elements A set of possible attribute names A set of possible attribute values A set of possible world states see also universe of discourse
mutable	The object, file, or resource can be modified. see also immutable
names	We change the name of things to better describe their role and function, and to bring it into the metaphor/paradigm.
criteria	Should convery functionality instead of implementation. Orthogonality. Embody a certain predictability in their names. Some kinds of names should have pairs: get & set, encrypt & decrypt, read & write.
	"The verbosity of all names should be proportional to the scope of the name." *Kirrily 'Skud' Robert, In Defense of Coding Standards http://www.perl.org/pub/2000/01/CodingStandard.html*
external source selecting	"In general, follow the languages conventions in variable name and other things." (Robert) "Whenever possible, name sets whatever they're called in the problem domain – whatever the customer calls them." (Kossuth) "Invent as little new terminology as possible" (ibid)

	"If at all possible, when inventing terminology, do not invent new acronyms" (ibid)
	If "using a tool… you might want to follow the tool's naming conventions" (ibid)
relationships	Use the proper verb phrase.
	"Don't name a binary relationship if you don't have to, or atleast don't define it separately from the set it connects." (Kovitz)
	"If you want must name a binary relationship, consider making it a noun, especially if the relationship is symmetrical." (ibid)
	"Consider converting any ternary relationship into named clauses." (ibid)
	"Naming a relation a verb or prepositional phrase… is most suitable when you want to speak of the relationship as a predicate, that is, as an expression that is either true or false." (ibid)
	"Functions or subroutine names should be verbs or verb clauses. It is unnecessary to start a function name with 'do'" (robert)
plurality	"make a set's name singular or plural according to what best applies to an *individual element* of the set." kovitz
	"the plurality of a variable name should reflect the plurality of the data it contains." robert
	Don't name "any set that you don't refer to elsewhere in the document." (kossith)
	When "the customer uses the same name for several different sets that you must distinguish [try to] find synonyms already in use. Do not call either set by the ambiguous word; avoid it entirely." kossith
notation	Often a skillful choice of reference system simplifies the work.
notation selecting	The choice of notation depends on:
	The kinds of problems you're trying to solve
	What environment you're trying to solve it in
	With whom you're trying to solve it
	How does the problem or task decompose into a given notation
	How easy is the problem to solve in the framework?
	How elegantly?
	Will it perform well?
notification	Notifications can be used to
numerical methods	Solving questions of valuation is better with (computer) analytic rather than symbolic method. Most realistic problems can't be solved analytically. There is no single method (or a small number of methods) that both suffices and is tractable. Each potential definition substituted for a given relation name requires a different method to solve – each is a different problem. Worse, descriptions involving differential equation are even more difficult than the rest: solutions of differential equation is a large of subfield of math.
object system	Object systems are 'tightly coupled': method base inheritance, events, delegates. Polymorphism, inheritance, encapsulation.
Function access to an object	In OOP a procedure can operate directly on an object if it is an instance method (ie an object has a table of allowed procedures). Otherwise it needs to employ an intermediary form.
OOP	an academically ideological way of manipulating globals.
	Tenants of polymorphism, encapsulation, inheritance

classification

Classified by:

How object is identified,

What and how its state is stored or associated with the object,

How the object is coupled with others

How actions are dispatched, the complexity of lookup

operator

Associativity – How to parenthesize with stream of same operators

Precedence – How to parenthesize a mix of different operators. The precedence of an operator should not depend on the types of potential operands.

Order of evaluation --

optimization

last call

The local call frame (activation record) will no longer be needed once the call is made, so a lot of resources can be cleaned up. It may not need to use a 'call' instruction, and may use a jump instead. Some architectures put all return values into a register or single memory area.

tail call

A further refinement of the last call optimization when the call is to the current procedure. The procedure can often be reworked into an iterative form, and may of the call fame structures will not need to be set up in subsequent calls.

dead store elimination

A=1; A=2;

loop unrolling

Reduces branching in a loop. Duff's device: increases index increment (or reduces the max), repeats the inside of the body to balance, and initially jumps into one of these redundant bodies so that the number of times executed are matched. Increasing the increment to 4:

```
max=(origMax + 3) / 4;
switch(origMax % 4)
{
  case 0: do{ inner;
  case 3: inner;
  case 2: inner;
  case 1: inner;}
  while (–Max);
}
```

static optimization

Predicated on the idea that repeating the optimization would be redundant and yield no change. The costs be incurred once.

static single assignment

Redundant expressions can be computed only once. Can be identified via Chow's algorithm. Partially redundant if redundant on some paths, but not all.

statistical optimization

works toward optimizing the common case(s) , was measured by profilers.

superoptimization

Converts an operation into a loop-free form.

order

Usually the number of parameters.

See also *rank*

parsers

A parser converts a sequence into another sequence:

$\text{Output}_j = \text{Parser}_{i,j}\ \text{Sequence}_i$

this involves:

lexical: turning it into words and symbols

parsing based on the syntax

resolving the named variables, functions, types, and other elements

semantic actions based on matching the patterns

Special cases of Parsers:

Top-down: LL(k)

Bottom-up: LR(k)

k = the amount we need to look ahead to distinguish between two or branches that we should take)

Objectives:

Minimize the amount we need to look ahead

Minimize backtracking

# of times we need to back track

Max depth we would back track

Average depth we would back track

Minimize the amount of state need to keep

Minimize work parser does. Backtracking, tests.

See also ATN, Chomsky hierarchy, Markov, regex, shift-reduce,

LALR(1) — An approximation to LR(1) parsing. *Frank DeRemer, MIT PhD thesis, 1969*

LR(k) — Bottom-up parser that became the definitive parsing solution (surpassing precedence methods). *Donald Knuth "On the Translation of Languages from Left to Right"* Information and Control, **8** *p607-639, 1965*

precedence — 1963 Floyd: operator precedence
1966 Wirth: simple precedence

static parsing — Take piece of text, determine its structure without executing it.

pattern matching — The process of examining a string to locate substrings or to determine if a string has certain properties.

predicate — It is a phrase posited to be either true or false. It includes atleast one variable, attribute or function; it may include an operator. There is often atleast one free (unbound) variable. Not all predicates are genuine properties.

see also *sentence*

problem solution search — Start with users knowledge of problem

Clear separation of constraints and combinatorial search

Discrete variables represent the primary decisions in the problem

High-level constraints represent the relationship between variables

Constraints can be combined to match the real-word's complex constraints

Generate multiple solutions quickly

Refine solutions

procedural semantics — The operations that one is supposed to carry out (rather than merely discussions of possible facts). Meaning that a statement takes action or changes the world. Backtracking can be very expensive (by throwing 'exception'), unreliable (errors reversible only by best effort) or not possible at all (as with destructive operations).

procedure form — Function *name*

Preconditions

Parameter conditions

Initial conditions

Routine

Post-conditions

propositional connectives Boolean operators (not, and, or, etc.) or set operators.

programming generic Represents efficient algorithms independently of data-representation. Interface to large set of data representations, and is flexible to choose appropriate representation.

language akin to a calculus with procedural semantics.

purity =Predicted length / Actual length *Maurice Halstead in 1970s*

Length $= n_1 + n_2$

Predicted length of well-written program $= n_1 \log_2(n_1) + n_2 \log_2(n_2)$

n_1 – number of unique operators

n_2 – number of unique operands

N_1 – total number of operators

N_2 – total number of operands

Minimal volume $= \log_2(n_1 + n_2)$ *bits*

Volume = information magnitude $= (N_1 + N_2)\log_2(n_1 + n_2)$

See also complexity

qualities brevity, clarity, simplicity.

latency, stringency

throughput

reliability, availability, performance, predicability

rank The rank of a formula is greater than or equal to the rank of each of its elements, operators, and parameters.

See also *order*

reducibility The reverse of composability, concerned with decomposing statements into observable terms.

reduction procedure Converts a declarative language into a procedural one.

see also *compiler*

reference A symbol may refer to something (usually this must be done thru a distinct meaning).

regular expression

Two regular expressions are equivalent if they recognize the same set of strings. Regular expressions can be differentiated using a set of rules analogous to Leibniz rules of differentiation. Given a regular expression R_1, the derivative (with respect to symbol 'a') is a regular expression R_2. R_1 recognizes the strings matched by R_2 when they are prefixed by 'a'.

See also Chomsky hierarchy, the method affine transforms for generating strings.

Summary: A description of how neurons behave, a pre-cursor of regular expressions
Warren McCulloch and Walter Pitts, "A logical calculus of the ideas imminent in nervous activity," Bulletin of Math. Biophysics *5 (1943) (reprinted in* Embodiments of Mind, *MIT Press, 1965)*

Table 8: Regular equivalences

	Equivalent to
a*	aa*
ØX	Ø
{empty string}X	X
(Ø|X)	
({empty string}|X)	

Summary: A regular expression compiler (targeting the GE-TSS machine), using an NFA.
Ken Thompson, "Regular expression search algorithm," Communications of the ACM *11(6),*

Table 9: Symbolic differentiation of regular expressions

	Equivalent to
$\frac{d}{da}b$	Ø (b ≠ a)
$\frac{d}{da}a$	{empty string}
$\frac{d}{da}a^*$	a*
$\frac{d}{da}a^+$	a*
$\frac{d}{da}XY$	$\left(\frac{d}{da}X\right)Y$
$\frac{d}{da}(X\|Y)$	$\left(\frac{d}{da}X\|\frac{d}{da}Y\right)$

Janusz Brzozowksi, "Derivatives of Regular Expressions" Journal of the Association of Computing Machinery, *V11N4 (October 164), p481-494*

relation algebra

Variables – properties of an entity – are compared. In CS this is used to specify sets of entities. In bulk, files of fixed-length records of multiple fields, which were selected and merged.

Summary: Relational DBs are a relabelling of existing practices promoting a pretense.
Henry Baker, letter to ACM, *Oct 15 1991, http://home.pipeline.com/~hbaker1/letters/CACM-RelationalDatabases.html*

Table 10: Regular to Relation translator

	Relational
fields	Column
files	Relations
merges	Joins
pointer	Key
records	Rows

S-plus syntax

The elements of the syntax is divided into:

Literals:

Numbers and complex numbers

Strings

Names

Commands

Functions – defined by assignment

Symbolic constants

Calls

Simple

Operations

Subscripting

Assignments

Conditionals

Loops & Flow of Control

While, for, repeat

Next, break, return

Grouping: braces & parenthesis

satisfaction
Carnap

The values a formula is true for; if true for the value or range of values. Or, rather, checking that a symbols value is consistent with the constraints.

See also resolution method, unification

Tarski

Every possible value for every variable in the universe, so long as the formula is true.

boolean

Givens:

A set of variables: $v_0,\ldots v_n$

A formula using those variables

Assign each variable a value (0,1) such that the formula evaluates to 1 – or find all such valid assignments. This is an NP complete task.

Steps:

"Decision step selects a variable for the next assignment, either statically with a fixed variable order, or dynamically, depending on information gathered during search.

"Deduction step infers information from the current partial assignment. Boolean constraint propagation… exploits the fact that a partial assignment can imply values for other variables.

"Diagnosis step analysis [a] contradictions' cause and uses the inferred knowledge to search more efficiently."

see also BDD (binary decision diagram), bounded model checking

Platzner, Marco "Boolean Satisfiability" IEEE Computer, *IEEE Computer, April 2000, p60*

Summary: based on binary Hyper-Resolution & Equality Reduction can solve many SAT problems without search. Bacchus "Exploring the Computation Trade of more Reasoning and Less Searching" 2002

parameter search problem

Givens:

Initial & boundary conditions

A set of constraints

Technique to solve the problem

Algorithm:

Starts by making an initial guess for the parameters

Calls the objective function & continues to adjust parameters to minimize the objective function. If the results are not satisfactory, repeats,

finds the best parameters with fewest evaluations.

Evaluating the objective function. Calls differential equation and compares

them with real data.

Differential Equation solver. Returns solution of ODE's for current guesses.

scene description graph

A hierarchical structure of nodes. Defines ordering of nodes.

Primary node: a group node, which may contain any number of other nodes, arranged in a hierarchical fashion.

Fields: parameters that modify nodes. There are zero or more fields. Provide the data to properly render scenes. Singly or multiple valued.

Self describing, a new node, but fully described.

Classification of node: shape, property (how shapes are drawn), group (collections of nodes)

Non-standard, all the fields are described first

Name for identification purposes

Lights, cameras, materials, textures

selector

Some identifier that refers to the method name so that no two identifiers refer the same method name, and an identifier specifies only one method name. With special discipline this can be where the selector is also a pointer to the method name.

shift-reduce parsing

BNF grammar is converted into a series of nodes like:

A link to the symbol table

Whether or not the item can be a null match

List of next states

The list of next states is made when checking the network

The symbol table is three parts:

The symbol (character) which is matched

The operation: Shift (which state to go to), reduce (number of items and which action to take), accept, error.

Hint: the extra bit of information for the operation

Then it builds a TRIE, assigning a number to each node first. The Trie is like the symbol table, except that shifts have state numbers, and there is a column for number. Then all of the symbols for next state are added, given a reduce step. Finally, all of the remaining symbols are added, and given an error state.

See also *parsing*

signature

Signatures are the list of types for a function's parameters, or a struct. Even assembly subroutines can be given signatures. This is very good at catching problems either at compile-time or run-time – like the wrong number of parameters, or parameter type mismatches. I accept the ability of a linker to choose from one of several different implementations of a subroutine with the same name, but different parameter sets.

software synthesis

see also compilation, path Pascal, System Generation

specification
ALGOL

Language specification tends to be divided along the lines of:

Structure of the language. Survey of the basic constituents, features.

Basic symbols, identifiers, number and strings, basic concepts. List of basic symbols, quantities and values.

Expressions. How they are formed (syntax), and their meaning. Variables (and subscripts), function designators, arithmetic, booleans expressions, designational expressions

	Statements: assignment, goto, dummy, for, procedure, compound statements and blocks.
	Declarations, usually including procedure, code bodies, scoping rules and influence of scopes, evaluate of inner expressions.
statements	Change program state
effect of	Change what executes next via control flow, dependencies, program slicing
string	a string is a sequence with an alphabet: lists, sets, arrays, streams, etc.
	operations: length, truncate, hash, append
	String canonicalization, paths, mapping trees, graph network to string.
	, anagrams and palindromes, parsing tasks,KWIC index, DNA Genetic algorithms
	Internal data structures and operations
	String attributes
	Turning special substrings into special characters
	Morpheme problem, akin to parsing
	Mapping string to extra without hash tables
alphabet	alphabet properties: does alphabet have an end-of-string character? Size of character representation (how many bits?)
	Small alphabet implementations. Bit and byte sized characters are the most common type of string; there are more fast algorithms available for these.
	There are some algorithms for large or unusual alphabets.
classification	topicalization, Classify strings, bayesian classification, ART1 classification, k-means. Each treats a document naively as a count of each type of word.
	Cross-referencing documents by words and phrases; uses a longest match in a table to build up list of documents.
matching	Complexity of string processing: Chomsky hierarchy
	string equality
	sort comparison
	String Comparison: Edit-distance (Levenshtein distance), Spell checking
	String & substring matching: KMP
	Prefix-matching, prefix tree:
	Suffix-tree
	Regex
	Phase structured (parsing: a string into a graph), LALR(1) shift-reduce
substring matching	Common sub-string identification: Karp-Rubin method, cross-reference of keyword and object
	Identifying keywords and phrases; concordance tables
	Applications:
	▪ cut-n-paste detection,
	▪ plagiarism detection,
	▪ compression tasks
	▪ finding files, documents, webpages, etc by keyword
	▪ finding documents / webpages with similar look or template (filters out normal

text, uses an alphabet of mark up / structuring elements)

structured programming

Control structures

Modular composition

Program format

Comments

Readability is more important than efficiency.

Stepwise refinement

Program verification

symbol

Constants, variables, types, fields, procedures, functions, programs, units, modules, libraries, and packages. Some have an identifier, a name or other means of identifying the symbol; typically such identifiers must be declared prior to use.

see also identifier

field

A storage location

property

Can be a field, have code associated with access or modification; can exhibit access control

System C
Metro II

Tagged signal model

Component: threads generate events; events are associated with required port interfaces. Port, who generated event (eg process), value set, set of tags.

Provided port receives processes and events from components with required ports.

Execution semantics. Revolve around the synchronization and execution of processes, based on event scheduling and annotation. Scheduler, constraint solver, annotation, mapping (relationship between events)

taint checking

Attempts to catch use of unvalidated values – i.e. inputs not range checked. Range checking is hard. It is not clear how to tell if a variable was range checked. Signed should have upper and lower range check. Unsigned should have at least upper-range checked. A destination might have flags indicate that it needs each of these.

task

"Any individual computation, set of computations, decision-making logic, or combination thereof that must be performed at run-time by software."

temporal logic

Temporal logic is, largely, the same as modal logic, except that it focuses more on the analytical needs of computer science. Primarily declarative statements used to validate the behaviour of various systems.

see also Büchi automaton, clock, modal logic, tense logic

Allen's interval algebra

X takes place before Y

X meets Y (one starts when the other ends)

X overlaps with Y

X starts Y (start of X == start of Y, duration of X < = duration of Y)

X during Y

X finishes Y

X is equal to Y

linear time

Linear time is represented a sequence of events (there is no concept of duration), augmenting prepositional logic with 8 operators describing the past and future:

Always after (in the future),

Sometime after (in the future),

Until

	Next cycle	
	Always in the past	
	Sometime in the past	
	Since	
	Previous cycle	
	This can be used to analyze contracts and behaviour of procedures or algorithms. This logic can be extended with counts or back-references.	
	See also Büchi automaton	
metric	Extends linear temporal logic with the concept of duration – each operator allows an upper and lower bound on duration.	
terms binding	Conversion of expressions and terms into immediately operational or evaluatable forms. Evaluation produces singular output in a specified range.	
tracing variables	tracks where a variable is assigned its value – the place in the code, and the call path.	
	It also tracks the origin the value – the previous variable and time of assignment. (which, in turn, may have its own origin of value).	
traits	A set of methods that are valid, implemented, and/or meaningful for an "object." For example, Apollo documentation suggested a regular file has the traits of open/read/write/seek, etc. while tty's have another set of traits. The term may come from LISP, Scheme, or Smalltalk. Is more flexible and harder to implement than Objective-C and Java interfaces.	
	see also interface	
translation emulators	"interprets program instructions at run time"	*Erik Altman, David Kaeli, Yaron Staffer, "Welcome to the opportunities of Binary Translation" IEEE Computer*
binary translation	"A set of techniques that directly translate compiled code" This can include profiling "to guide optimization"	
dynamic translator	"translates between the legacy and the target ISA, caching the pieces of code for future use." May be integrated with the VMM: if a jump or access is made to a region of memory that has not been translated, this can be trapped and passed back to the special interpreter / translator.	*Cindy Zheng, Carol Thompson "PA-RISC to IA64: Transparent Execution, No Recompilation"* IEEE Computer *March 2000, p47-52*

	if (Native Code exists for VAddress)
	{
	if address that called us is native, update to directly call the Native Code for the VAddress.
	jump to that native code
	}
	if (the execution count for the VAddress > translation threshold)
	{
	translate the block, and update the mapping. update known translated callers to jump directly to it.
	jump to the native code
	}
	interpret code to branch
	if there is native code for the branch target
	set branch to jump to the native code
	else
	redirect branch to the first step above	

static translators	"translates offline and can apply more rigorous code optimizations than" any other (altman)	*Altman et al,* ibid
dynamic optimization	Optimization issues: dead code elimination, address translation reduction, memory aliasing reduction.	
	"ISA remapping – handle register overlaps present in the legacy ISA and remap to the target ISA."	

	"basic block reordering – keep the target image execution as sequential as possible so that conditional branches will typically fall through, which helps speed instruction fetching and cache performance."
	"memory coloring – improve the mapping of the translated image onto the memory hierarchy of the target environment."
	"code specialization – clone procedures based on the invariance of parameter values."
possible issues	"the new architecture has fewer registers than some. legacy register values must be kept in memory with costly loads and stores used to access them."
	"system states, stored in special-purpose registers."
	"memory mapped IO… references to IO locations can have side effects.. and must be done in program order and without caching."
	"instructions that must execute atomically with respect to memory"
types of things in a language	Operators.
	Statements: control and declaration
	Blocks
	Functions & procedures
	Modules
	Signals & exceptions
	Messages & events
	Handlers
	Variables
	Identifiers
	Defaults
types	Constrain what can say about it's interface and how to ensure compatibility.
dynamic	Item stored in slot has a specific type associated with the item.
static	Memory slot has a specific type associated with it; sound systems ensure that only items of the same type are inserted into the slot.
latent-typing	The types that a slot may have, based on knowledge about the possible types that may set it.
parameterized types	Write code once, but several types of arguments: eg templates.
	See also widening.
partial ordering	An object of type A can be converter to one of type B without loss of information. (B is "wider" than A, or A is "less than" B)
safety	Promotion to a wider, at least as accurate type, can be done automatically – so long as there is only one conversion technique.
strictness	Whether a type can be treated as another; auto-conversion
strong types	Emphasis on catching errors as soon as possible, e.g. with the compiler; however it makes modularity/components/reuse more difficult.
weak	Safety is only possible with static analysis and extensive run-time checking. This checking is done at the last possible moment, so the system may exhibit incorrect behaviour only after running for extended period of time (after the actual violation).
types of small data people like	Numbers, dates, arrays (indexed and associative), patterns, text and word
undefined access	If using a slot that is not defined. Trapping this is difficult.

unification

Unification is a key step in the resolution method, operating like regular expression matching. Unification operates on a *substitution* table (see the example below) adding further entries as it binds variables. Unification takes this table, a goal clause, and a clause in the table. It tries every combination of variable assignments to make the two clauses equivalent. It steps thru the both clauses in the same way:

John A Robinson "A machine oriented logic based on the resolution principle." Journal of the ACM *12(1):23-41 January 1965, Syracuse University*

If this element is a *free* variable, *bind* it to the corresponding element in the other clause. This is done by adding an entry into the substitution table.

If this element is a bound variable, look up its value; if it is a *literal*, use that. Perform the same on the other side. If the two values are defined, but do not match, abort; unification cannot be performed.

If the element has *parameter* or sub-ordinate elements, a unification step is performed on those parameter clauses of both main clauses.

This process repeats until no more items are added to the table.

This process effects the inference of variables values (or sets of acceptable values). It can link variables together, showing those that alias each other. It can be modified to remove possibilities from a potential set.

Term Rewriting systems perform a string substitution, replacing each occurrence of a variable with its bound value.

It is easy to understand the substitution table in cases where a variable can be bound to a simple value (e.g. a scalar or a string), a structure whose elements are found. What makes unification powerful is the ability it for a variable to be bound to another variable – v_4 (in this context) will inherit whatever v_1 is bound to. A variable can also be bound to a structure, whose elements might not be bound, or might be bound to another variable.

One drawback is that the table can have cycles. An *occurs check* operation can be attempted to catch this occurrence, but the check is very expensive.

see also resolution principle, tableau

Table 11: *Example substitution table*

Variable	*Binding*
v_1	1
v_2	"bob"
v_3	house(red)
v_4	v_1
v_5	house(v_2)

valuation function

In theories constructed as a *model*, one needs to know how names and terms refer to entities and their properties, and how to evaluate sentences. For example Sally's height & mass, or an electrons charge. This is called a 'valuation function' although it is seldom a simple function, and often better understood as a procedure. This valuation assigns value for formula based on those references and how they combine (composition), table of forms and their values (idiomatic).

see also evaluation procedure

method1

One method is to use the problems declarative specification to specify a grammar and a family of automatons. The first automaton is special in that the sentences it recognizes (accepts) are also solutions to the problem. The other, optional, automatons generate fragments of the language that may be present in the acceptable sentence(s). Despite the unusual pretense of the solution as a sentence in an imaginary language, this technique can be very efficient.

see also Chomsky hierarchy, language fragment

variable

Value of the variable is controlled by a quantifier, is a parameter or is a constant

bound

free

A variable that is not a constant, not a parameter, and is not controlled by a quantifier

visual

Hypercard mid 1980s

programming systems

Labview

NextStep Interface Builder 1988

Visual Basic 1991

WalkSAT

```
for(I=1; I < Max Tries; I++)
{
  solution = random truth assignment
  for (J=1; J < MaxFlips; J++)
  {
    if all clauses satisfied clause then return solution
    c ← random unsatisfied clause
    with probability p
      flip a random variable in c
    else
      flip variable in c that maximizes the number of satisfied claims
  }
}
return failure
```

max WalkSAT

```
for(I=1; I < Max Tries; I++)
{
  solution = random truth assignment
  for (J=1; J < MaxFlips; J++)
  {
    m = sum of weights(sat clauses)
    if m > threshold then return solution
    c ← random unsatisfied clause
    with probability p
      flip a random variable in c
    else
      flip variable in c that maximizes m
  }
}
return failure with best solution found
```

a version without memory explosion is at http://alchemy.cs.washington.edu

witness function

A function that 'testifies' a proposition is highly likely to be true.

see also probability estimator