

# **“The fault detectability domain: Attack in depth”**

by

Les Hatton

The Computing Laboratory, University of Kent  
L.Hatton@ukc.ac.uk, lesh@oakcomp.co.uk

Version 1.2: 17/Oct/2002

# Overview



---

## Principles

▣ Technical input

▣ Implementation policy

▣ Architecture

▣ Status

▣ Future direction

# Technologies



---

## **The following technologies are available for detecting defects**

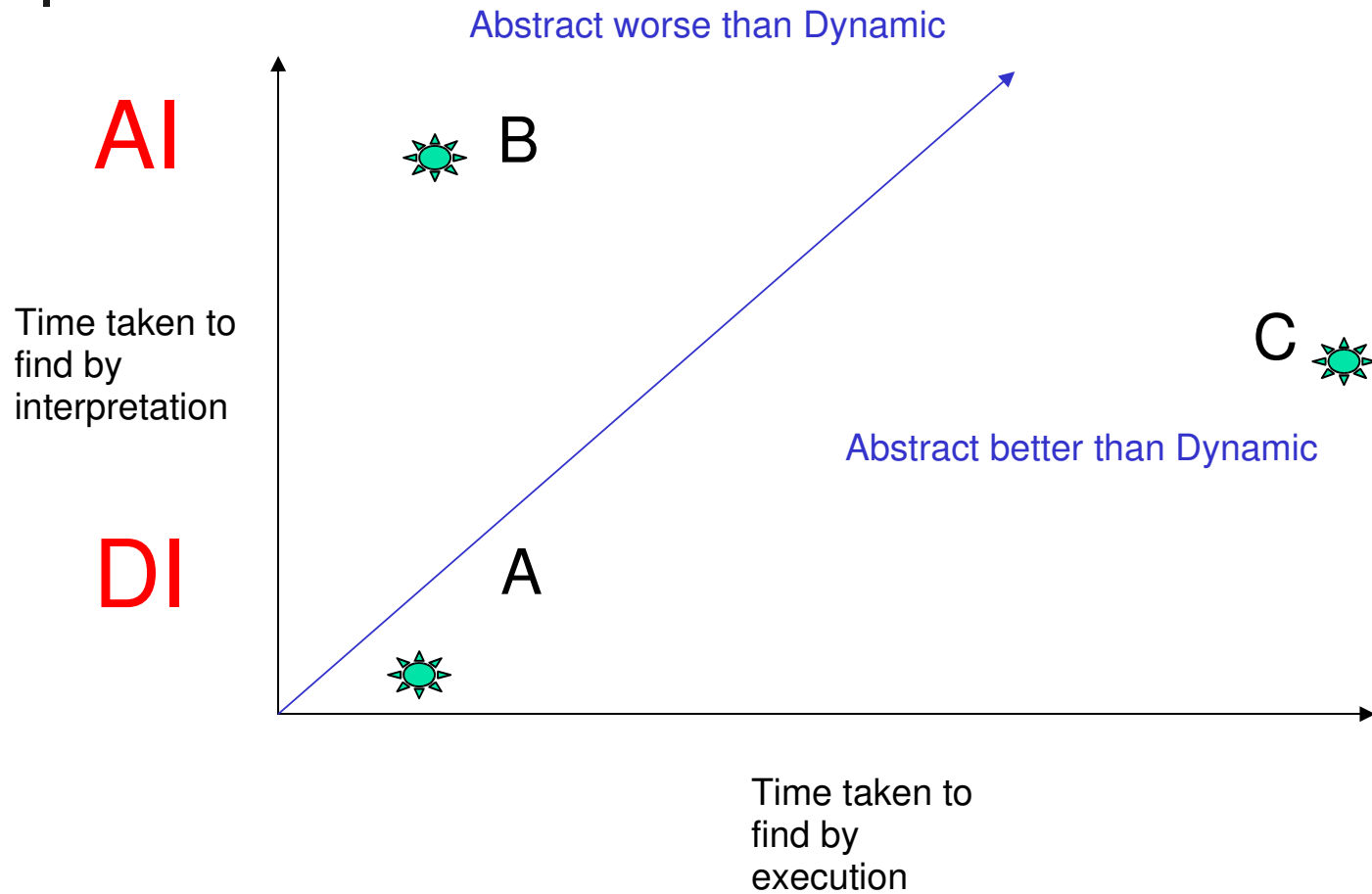
- **Direct interpretation (DI)**

A program is checked for strict linguistic conformance and freedom from a wide class of related faults.
- **Abstract interpretation (AI)**

Linear relationships are extracted or approximated from a program and linear algebra used to infer equalities or inequalities which could lead to failure.
- **Abstract execution (AE)**

The program is run in parallel with the defined abstract model of computation. Failing faults are detected directly.

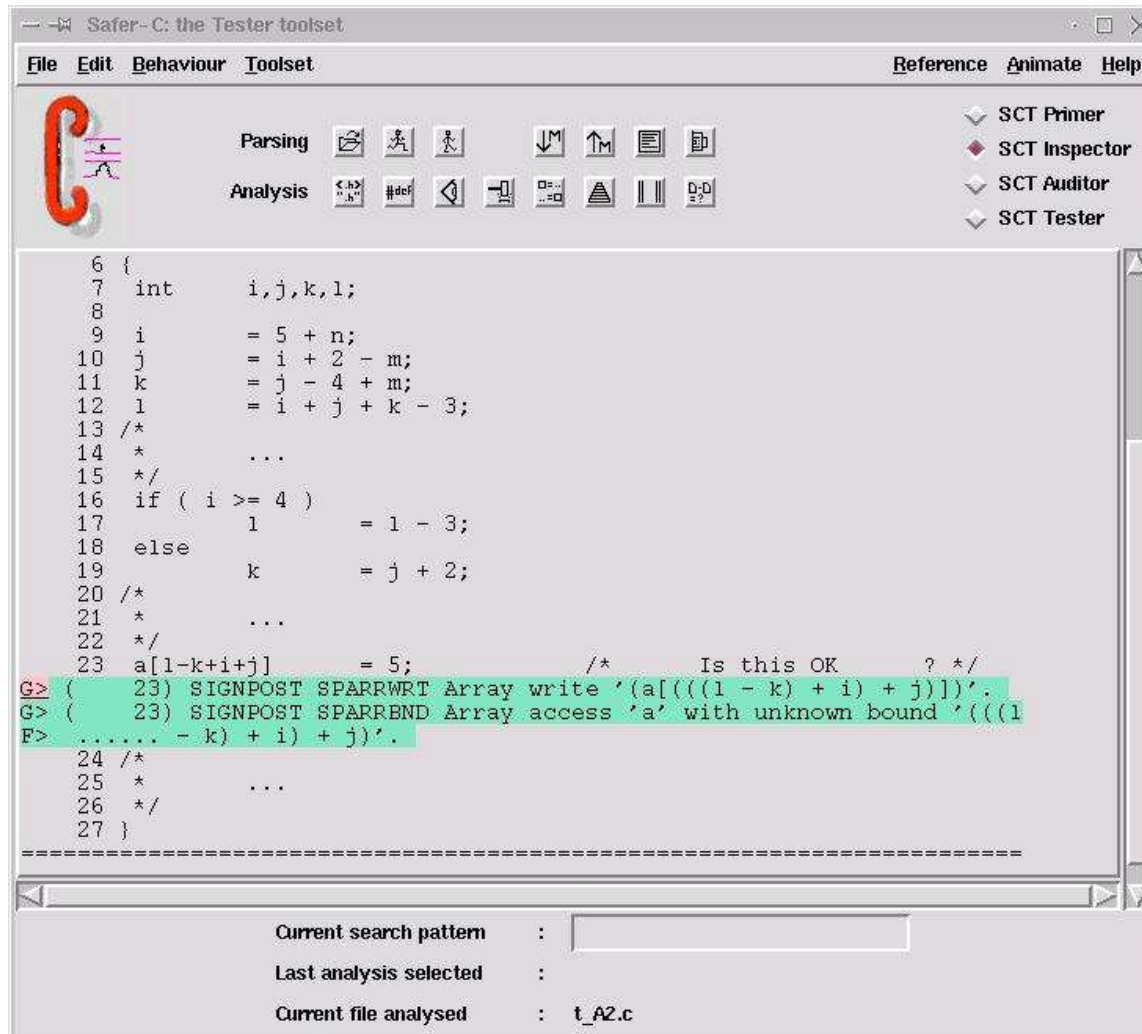
# The detectability domain



# Direct interpretation - single component

*Advantage*  
Very fast

*Disadvantage*  
Onus on user to  
Inspect.



The screenshot shows the 'Safer-C: the Tester toolset' application window. The interface includes a menu bar (File, Edit, Behaviour, Toolset, Reference, Animate, Help), a toolbar with icons for parsing and analysis, and a list of tool components (SCT Primer, SCT Inspector, SCT Auditor, SCT Tester). The main window displays a C code snippet with the following content:

```
6 {
7  int    i,j,k,l;
8
9  i      = 5 + n;
10 j     = i + 2 - m;
11 k     = j - 4 + m;
12 l     = i + j + k - 3;
13 /*
14 *      ...
15 */
16 if ( i >= 4 )
17     l    = 1 - 3;
18 else
19     k    = j + 2;
20 /*
21 *      ...
22 */
23 a[l-k+i+j] = 5;          /*      Is this OK      ? */
G> ( 23) SIGNPOST SPARRWRT Array write '(a[(((1 - k) + i) + j)])'.
G> ( 23) SIGNPOST SPARRBND Array access 'a' with unknown bound '(((1
E> ..... - k) + i) + j)'.
24 /*
25 *      ...
26 */
27 }
```

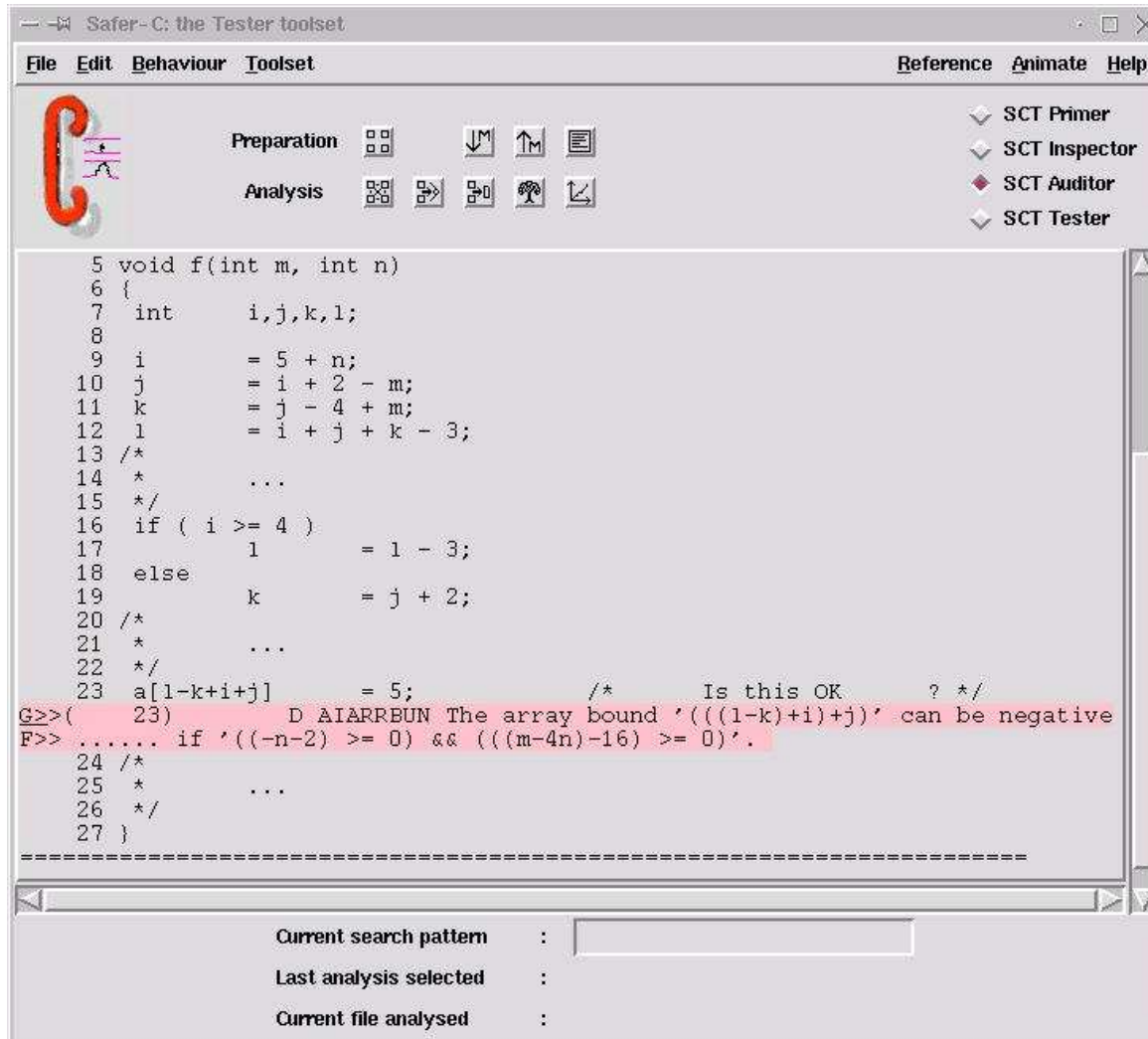
Below the code, the analysis results are displayed:

```
Current search pattern : 
Last analysis selected : 
Current file analysed  : t_A2.c
```

# Abstract interpretation - single component

*Advantage*  
Almost as fast

*Disadvantage*  
onus on user to  
inspect but  
more information  
provided.



The screenshot shows the 'Safer-C: the Tester toolset' window. The menu bar includes 'File', 'Edit', 'Behaviour', 'Toolset', 'Reference', 'Animate', and 'Help'. The toolbar has 'Preparation' and 'Analysis' sections with various icons. On the right, a list of tools is shown: 'SCT Primer', 'SCT Inspector', 'SCT Auditor', and 'SCT Tester'. The main text area contains a C program with a function 'f' that calculates values for variables 'i', 'j', 'k', and 'l'. A comment on line 23 asks 'Is this OK ? \*/'. Below the code, the analysis output shows a warning: 'D AIARRBUN The array bound '((1-k)+i)+j' can be negative'. The warning is highlighted in pink. The output also shows the condition 'if '((-n-2) >= 0) && ((m-4n)-16) >= 0''. At the bottom, there are fields for 'Current search pattern', 'Last analysis selected', and 'Current file analysed'.

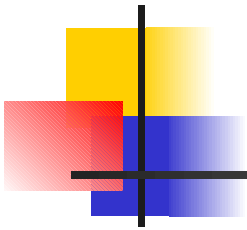
```
5 void f(int m, int n)
6 {
7     int    i,j,k,l;
8
9     i      = 5 + n;
10    j      = i + 2 - m;
11    k      = j - 4 + m;
12    l      = i + j + k - 3;
13    /*
14     *      ...
15     */
16    if ( i >= 4 )
17        l      = 1 - 3;
18    else
19        k      = j + 2;
20    /*
21     *      ...
22     */
23    a[1-k+i+j] = 5;          /*      Is this OK      ? */
G>>( 23) D AIARRBUN The array bound '((1-k)+i)+j' can be negative
F>> ..... if '((-n-2) >= 0) && ((m-4n)-16) >= 0'.
24    /*
25     *      ...
26     */
27 }
```

Current search pattern :

Last analysis selected :

Current file analysed :

# Abstract interpretation - system wide



*Advantage*  
Explicit detection

*Disadvantage*  
Took many hours

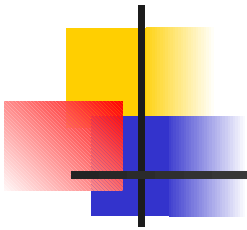
```
5 void f(int m, int n)
6 {
7   int    i,j,k,l;
8
9   i      = 5 + n;
10  j      = i + 2 - m;
11  k      = j - 4 + m;
12  l      = i + j + k - 3;
13 /*
14 *      ...
15 */
16 if ( i >= 4 )
17     l    = 1 - 3;
18 else
19     k    = j + 2;
20 /*
21 *      ...
22 */
23 a[1-k+i+j] = 5; /* Is this OK ? */
G>>( 23) D AIARRBAC The array bound '((1-k)+i)+j' has the illegal
F>> ..... value '(-1)'.
24 /*
25 *      ...
26 */
27 }
```

Current search pattern :

Last analysis selected :

Current file analysed :

# Abstract execution - system wide



*Advantage*  
Explicit detection

*Disadvantage*  
Much slower than  
normal real time  
But faster than  
Abstract Interpretation

```
4 */
5 void f(int m, int n)
6 {
7     int    i,j,k,l;
8
9     i      = 5 + n;
10    j      = i + 2 - m;
11    k      = j - 4 + m;
12    l      = i + j + k - 3;
13    /*
14     *      ...
15     */
16    if ( i >= 4 )
17        l      = 1 - 3;
18    else
19        k      = j + 2;
20    /*
21     *      ...
22     */
23    a[1-k+i+j] = 5;          /*      Is this OK      ? */
G>>( 23) D STABOUND Array bound violation. Array 'a' accessed
F>> ..... with index '(-1)', minimum allowed is '0'. A 'one-off'
F>> ..... error.
24    /*
25     *      ...
26     */
27 }
```

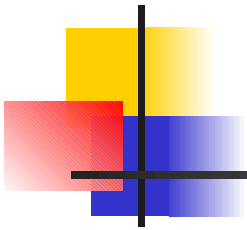
Current search pattern :

Last analysis selected :

Current file analysed :



# Direct execution – assertion defence



*Advantage*  
Very fast

*Disadvantage*  
Programmer  
must annotate.

View selected file check/Polyhedra/t\_A2.c

```
6: {
7:     int    i,j,k,l;
8:
9:     i      = 5 + n;
10:    j      = i + 2 - m;
11:    k      = j - 4 + m;
12:    l      = i + j + k - 3;
13: /*
14:  *      ...
15:  */
16:     if ( i >= 4 )
17:         l      = l - 3;
18:     else
19:         k      = j + 2;
20: /*
21:  *      ...
22:  */
23:     assert( (l-k+i+j) >= 0 );
24:     a[l-k+i+j] = 5;          /*      Is this OK      ? */
25: /*
26:  *      ...
27:  */
28: }
```

OK

# Technical input



---

## **The following sources were used for version 1.1:-**

- The April 1998 MISRA document, ISBN 0-9524156-9-0
- The (undated) Technical Clarification – Draft one, authored by Gavin and Paul
- The existing conformance suites provided by Andrew Burnard, Derek Jones and Les Hatton
- Ongoing input from the steering committee

# Implementation policy



---

## **The following simple policy was used:-**

- Hierarchy of wording
  - Bold face part of rule assumed normative overriding any commentary
  - Commentary used if bold face ambiguous
  - False positives favoured if commentary ambiguous; query files produced; rationale modified.
- No dependence on file inclusion
- Required rules done first

# Architecture: Goals



---

## Goals:-

- Orthogonality, (no crosstalk but *not* 1 item 1 file).  
I failed :-)
- Portability. (The architecture should run on any platform)
- Verifiability. There must be a way of detecting whether the source of a rule was altered. There must also be no dependence on inclusion, so parts of the C standard headers had to be constructed.

# Architecture: Construction



---

## **Construction:-**

- Perl was used for the engine. This should run without change on Windows, Linux, Unix and so on although are still two shell scripts in 1.1.
- An automatic checksum is inserted into the conformance suite on a file by file basis so that source change can be detected.

# Architecture: Deliverables



---

**All deliverables are open source, released under the GPL:-**

- Readme + Rationale (html) + GPL licence
- MISRA conformance suite, (completely self-contained)
  - Every rule has a positive file (must detect) and a negative file (must not detect)
  - Some rules have query files (not part of conformance but may be in future)
- Conformance suite architecture

# Status



---

## **Version 1.1c available now:-**

- 497 test cases for 45 rules in 90 files. (40 days work so far in 46 CRs). (Milestone 15-10-2001, 23:59:00)

## ***Version 1.1...***

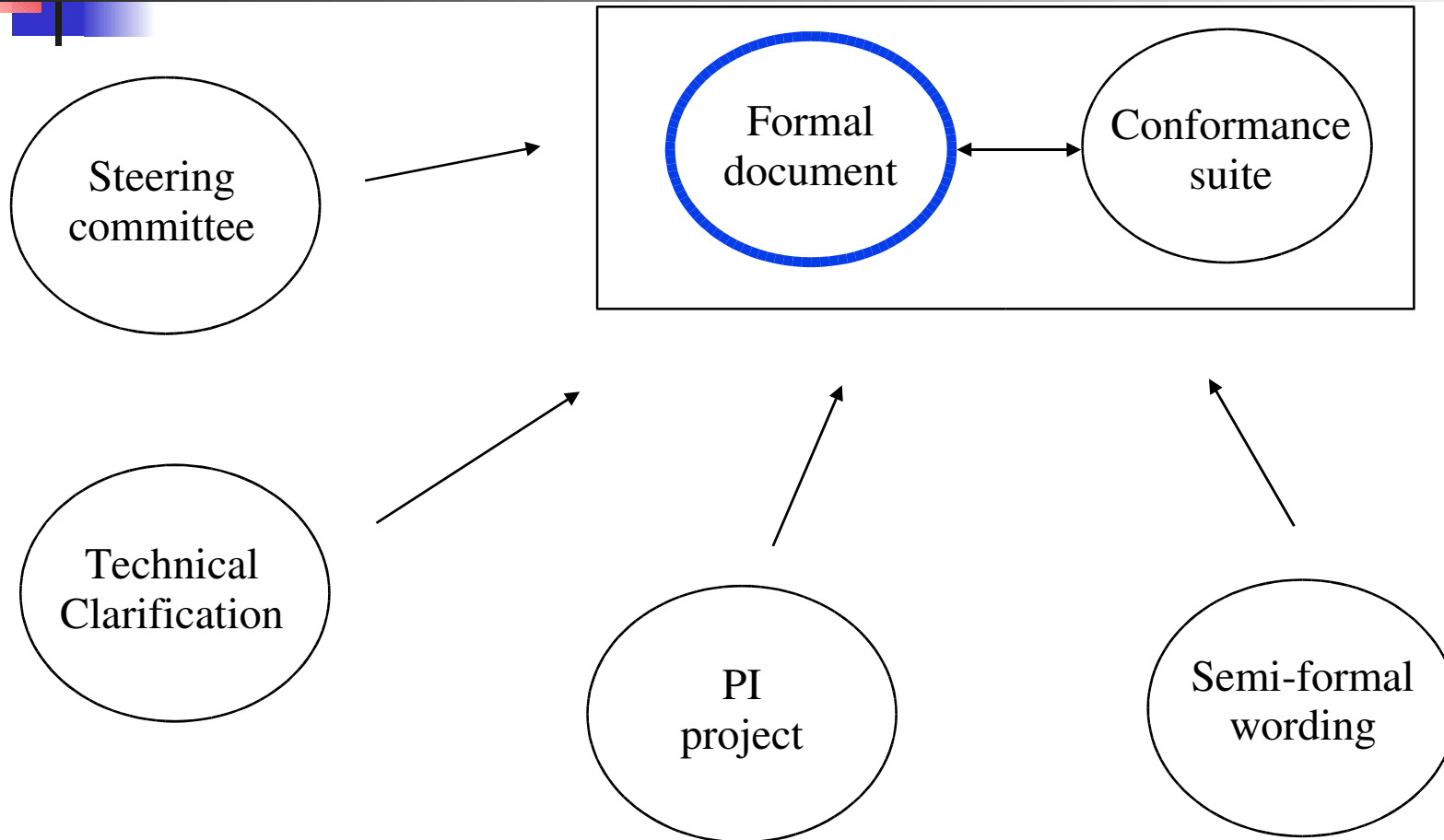
- Remaining required rules a few at a time

## ***Version 2.0***

- Advisory rules also a few at a time

Later versions to assimilate revised wording as it develops

# Future direction: Sources of change





# Availability



---

## **Download sites:-**

- The conformance suite itself is available at the UKC national Safer Language Subsets project site:-

<http://www.cs.ukc.ac.uk/national/SLS>