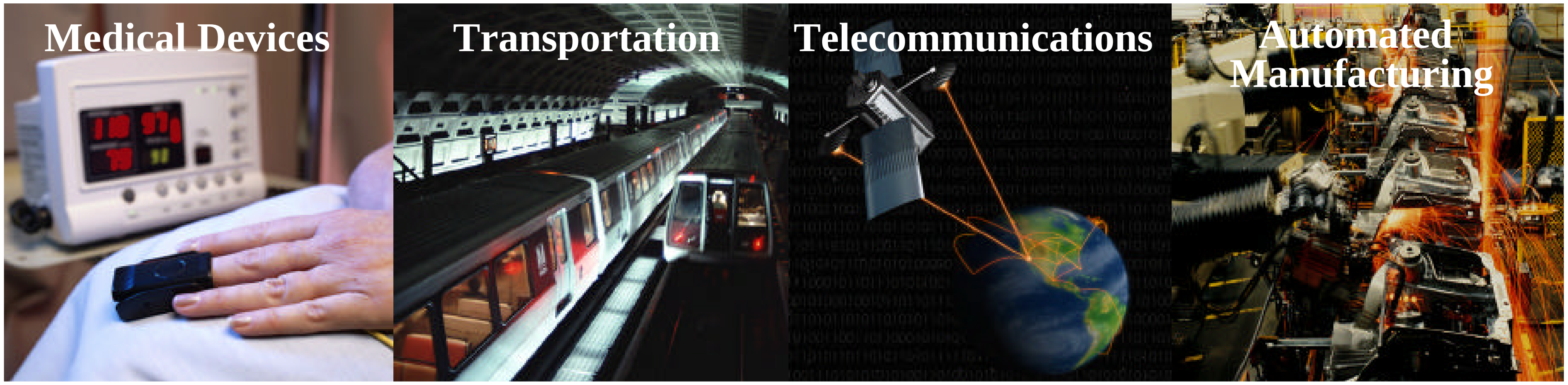
Parnas Tables: A Practical Formalism

Joanne M. Atlee
Department of Computer Science
University of Waterloo

Critical Software



Software is increasingly used to control or manage **critical systems**

safety-critical systems in which a failure can lead to loss of life
(e.g., medical devices, nuclear power plants, airplanes, cars, trains)

mission-critical systems in which a failure can cause significant loss of property
(e.g., spacecraft, satellites, manufacturing, security systems, financial systems)

How to Achieve Confidence in Critical Software?

There are several **complementary** verification activities.

1. **Review software documents** (requirements, design, code)
   - to **reveal errors early** in the development process, when they are easier to correct (cf. testing, code reviews)
   - to **exhaustively examine** an artifact (cf. testing)
   - to **locate defects** (cf. testing)

2. **Test code systematically** to confirm expected behaviour
   to **evaluate the final product** in its operational environment (cf. reviews)

3. **Test code randomly** to reveal unexpected behaviour
   to help assess the **software's reliability** (cf. reviews)

4. **Perform hazard analysis** to detect and avoid causes of failures

This talk focuses on writing and reviewing software documentation

Software Documentation

Software Documentation - technical documents that explain a software system's

- **requirements** - required goals of system
- **specification** - specified functionality of system
- **design** - decomposition of system into modules, and
  - specified functionality of each module
- **descriptions** - actual functionality of program fragments

(Im)Precise Documentation

If one does not have a **precise** definition of a system’s desired behaviour, how can one possibly expect to **evaluate** that the implemented system meets its requirements?

Mathematical Documentation

In other engineering disciplines, “precise documentation” means **mathematical definitions**

- **unambiguous**
- **consistency, completeness**, and other desired properties are well-defined and can be checked
- **composition** of components is well-defined

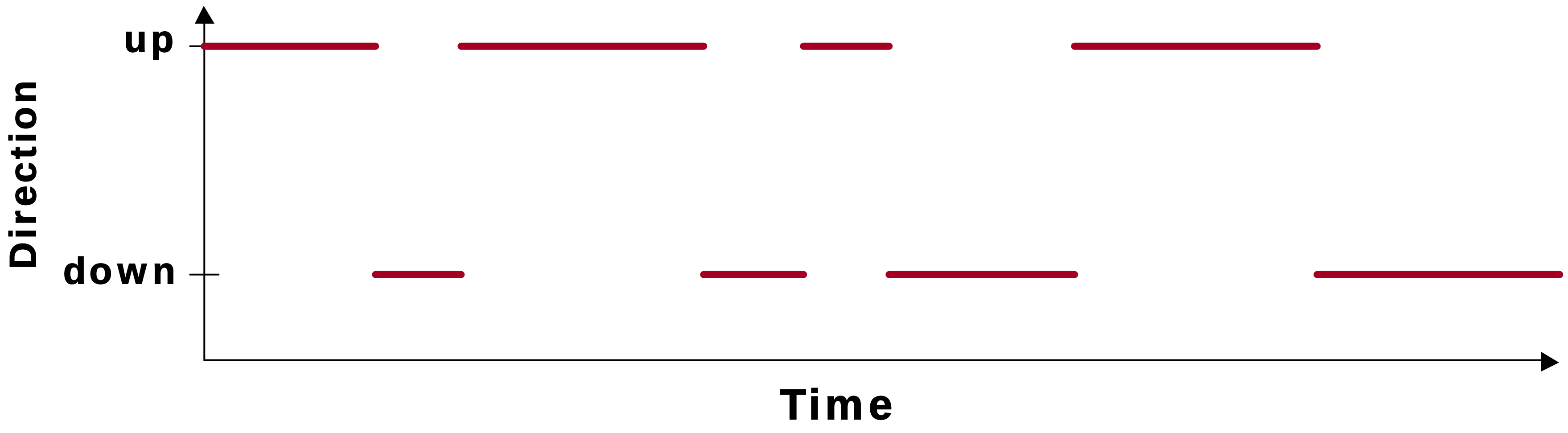
Mathematical Documentation

In contrast, mathematical methods are not widely used to document software because software can implement a function

- that has **many discontinuities**
- whose **domain and range are tuples of distinct types**

making it difficult to express behaviour in a **compact** mathematical definition.

Example: Elevator Direction



Elevator Example

An elevator’s direction depends on its **current direction (dir)**, the **floor** that it is on **(loc)**, and what **requests** are pending **(Req[]).**

It travels in a given direction until

- there are **no** more pending **requests** in the **current direction**
- and there are pending **requests** in the **opposite direction**.

dir : {Up, Down}
loc: {1..n}
Req[1..n]: boolean

$$\mathrm{ElevDir}(dir, loc, Req[\,]) = \begin{cases} \mathrm{Up} & (dir = Up \wedge \exists\, f.(f \geq loc \wedge Req[f])) \vee \\ & (dir = Down \wedge \neg \exists\, f.(\leq loc \wedge Req[f]) \wedge \\ & \quad \exists\, f.(f > loc \wedge Req[f])) \\ \mathrm{Down} & (dir = Down \wedge \exists\, f.(f \leq loc \wedge Req[f])) \vee \\ & (dir = Up \wedge \neg \exists\, f.(f \geq loc \wedge Req[f] \wedge \\ & \quad \exists\, f.(f < loc \wedge Req[f])) \\ dir & \text{otherwise} \end{cases}$$

Practical Formalisms

Practical Formalisms are notations with **precise semantics** that can be **read and reviewed** by domain experts and software professionals.

- They have a **formal, mathematical model**
- They encourage the use of **separation of concerns** and **abstraction** to decompose and simplify a problem
- They have **diagrammatic constructs** for expressing functions and relations
- …that encourage the writer to **consider completeness**

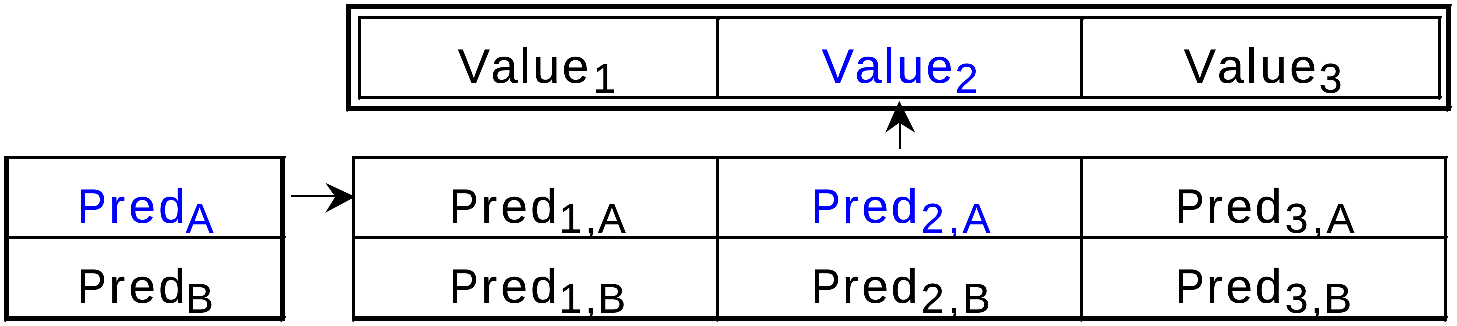
Examples: Statecharts, SDL, Petri-Nets,
Parnas Tables, SCR, CoRE, RSML, Tablewise

Parnas Tables

Parnas Tables use **tabular** constructs to organize mathematical expressions, where

- rows and columns separate an expression into **cases**
- each table entry specifies either the **result value for some case** or a **condition that partially identifies some case**

Example: Inverted Table



$Value_1$	$Value_2$	$Value_3$

$Pred_A$
$Pred_B$

$Pred_{1,A}$	$Pred_{2,A}$	$Pred_{3,A}$
$Pred_{1,B}$	$Pred_{2,B}$	$Pred_{3,B}$

$$F_{2,A} \equiv \text{if } Pred_A \wedge Pred_{2,A}$$
$$\text{then Result} = Value_2$$

$$F \equiv \bigcup_{j\equiv A..B,\ i=1..3} F_{i,j}$$

Inverted Table

ElevDir(dir,loc,Req[]) =

	Up	Down
dir=Up	$ f.(f³loc Ù Req[f])	Ø $ f.(f³loc Ù Req[f]) ^ $ f.(f<loc Ù Req[f])
dir=Down	Ø $ f.(f£loc Ù Req[f]) Ù $ f.(f>loc Ù Req[f])	$ f.(f£loc Ù Req[f])

Up	(dir = Up Ù $ f.(f³loc Ù Req[f])) Ú (dir = Down Ù Ø $ f.(f£loc Ù Req[f]) Ù $ f.(f>loc Ù Req[f]))
Down	(dir = Down Ù $ f.(f£loc Ù Req[f])) Ú (dir = Up Ù Ø $ f.(f³loc Ù Req[f] Ù $ f.(f<loc Ù Req[f]))
dir	otherwise

Multiple Table Types

The term **Parnas Tables** actually refers to a collection of **table types** and **abbreviation strategies** for organizing and simplifying functional and relational expressions.

An expression can usually be represented in several table types. The documenter’s goal is to choose (or create) a table format that produces a **simple, compact representation** for that expression.

Example: Normal Table

	$Pred_1$	$Pred_2$	$Pred_3$
$Pred_A$	$Value_{1,A}$	$Value_{2,A}$	$Value_{3,A}$
$Pred_B$	$Value_{1,B}$	$Value_{2,B}$	$Value_{3,B}$

$$F_{2,A} \equiv \text{if } Pred_A \wedge Pred_2$$
$$\text{then } Result = Value_{2,A}$$
$$F \equiv \bigcup_{j=A..B}^{j=1..3} F_{i,j}$$

Normal Table

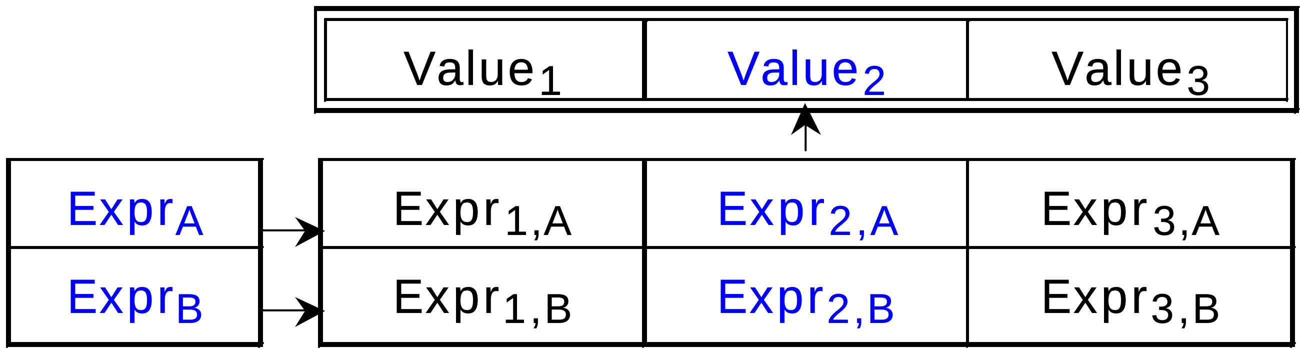
ElevDir(dir,loc,Req[]) =

		$ f.(f£loc Ù Req[f])	
		true	*false*
$ f.(f³loc Ù Req[f])	*true*	dir	Up
	false	Down	dir

Up	(dir = Up Ù $ f.(f³loc Ù Req[f])) Ú (dir = Down Ù Ø $ f.(f£loc Ù Req[f]) Ù $ f.(f>loc Ù Req[f]))
Down	(dir = Down Ù $ f.(f£loc Ù Req[f])) Ú (dir = Up Ù Ø $ f.(f³loc Ù Req[f] Ù $ f.(f<loc Ù Req[f]))
dir	otherwise

Decision Table

A **Decision Table** is useful for representing a function or relation whose **domain is a tuple** (possibly of distinct types). One dimension of the table itemizes the elements of the domain tuple.



$F_2 \equiv$ if $Expr_A = Expr_{2,A}$
and $Expr_B = Expr_{2,B}$
then Result = $Value_2$

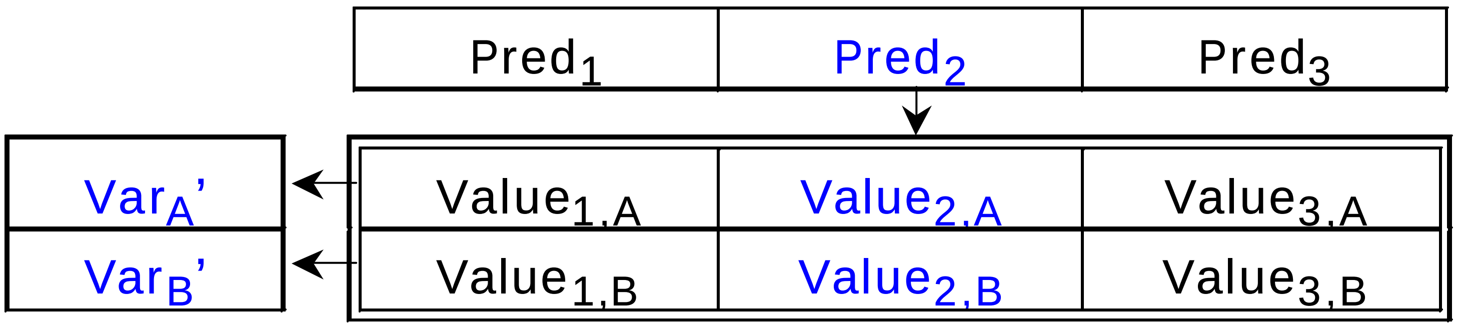
$F \equiv \bigcup_{i=1..3} F_i$

ElevDir(dir,loc,Req[]) =

	Up	Down	Down	Up
dir	Up	Up	Down	Down
$ f.(f³loc Ù Req[f])	true	false	---	true
$ f.(f£loc Ù Req[f])	---	true	true	false

Vector Tables

A **Vector Table** is useful for representing a function or relation whose **range is a tuple** (possibly of distinct types). One dimension of the table itemizes the elements of the range tuple.



$F_{2,A} \equiv \text{if } \text{Pred}_2$

$\quad \text{then } \text{Var}_A' = \text{Value}_{2,A}$

$$F \equiv \bigotimes_{i=A}^{B} \bigcup_{j=1}^{3} F_{i,j}$$

	Req[loc]	¬Req[loc]			
		$\neg\exists f.\text{Req}[f]$	$\exists f.(f>loc \wedge \text{Req}[f]) \wedge \neg\exists f.(f<loc \wedge \text{Req}[f])$	$\exists f.(f<loc \wedge \text{Req}[f]) \wedge \neg\exists f.(f>loc \wedge \text{Req}[f])$	$\exists f.(f<loc \wedge \text{Req}[f]) \wedge \exists f.(f>loc \wedge \text{Req}[f])$
dir'	dir	dir	Up	Down	dir
speed'	idle	idle	moving	moving	moving

Properties of Parnas Tables

For each table type, there are rules for identifying

- **distinct cases (subfunctions, subrelations)**
- **mission cases (incompleteness)**
- **conflicting cases (inconsistency)**

	Up	**Down**	**??**	Down	**Up**	**Down**
dir	Up	**Up**	**Up**	Down	**Down**	**Down**
∃ f.(f≥loc ∧ Req[f])	true	**false**	**false**	---	**true**	**true**
∃ f.(f≤loc ∧ Req[f])	---	**true**	**false**	true	**false**	**false**

A-7E Experience

A-7E U.S. Naval Aircraft:

Onboard flight software for an operational naval aircraft (navigation, navigational update, weapons delivery)

Project:

An experiment, funded by the Naval Research Laboratory (NRL), to evaluate state-of-the-art software engineering methods

Experience:

- Introduced the **first Parnas Tables** (without formal definition) in the Software Requirements Specification (SRS)
- SRS was **reviewed by domain experts, pilots**, who found hundreds detail errors

A-7E Experience

Since Then:

- The software manager for the **A-7D Air Force aircraft** had his team modify the A-7E document to reflect the A-7D requirements.

  This became the **living document** of A-7D software behaviour.

- NRL continues to study the use of Tables in documenting software requirements and specifications (**SCR method**), including methodology and tool support.

Darlington Experience

Darlington nuclear shutdown system:

Two independent systems, each of which is responsible for shutting down the nuclear reaction in the event of an accident.

Project:

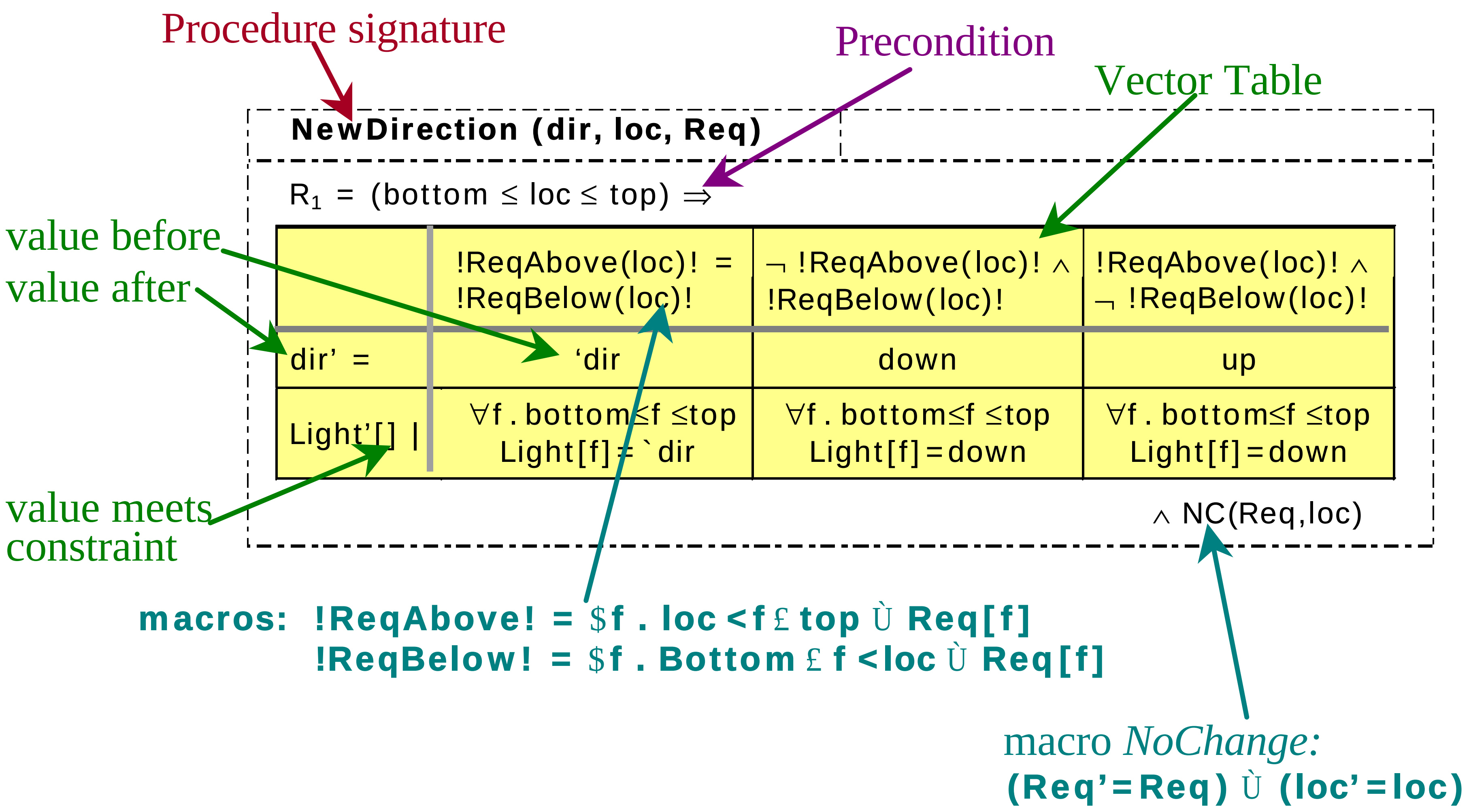
To determine whether the already-developed software and documentation met standards and could be certified.

Experience:

- Introduced **program-function tables** for documenting code
- Defined and executed a **systematic inspection process**
- 35-person-years task; relatively few important discrepancies found; but **gained confidence in the code**

Program Function Tables

A **Program Function Table** is an annotated Mixed Vector Table that describes the behaviour of a **procedure** or a **sub-procedure**.



NewDirection (dir, loc, Req)

R_1 = (bottom ≤ loc ≤ top) ⇒

	!ReqAbove(loc)! = !ReqBelow(loc)!	¬ !ReqAbove(loc)! ∧ !ReqBelow(loc)!	!ReqAbove(loc)! ∧ ¬ !ReqBelow(loc)!
dir’ =	‘dir	down	up
Light’[] |	∀f . bottom≤f ≤top Light[f]≠`dir	∀f . bottom≤f ≤top Light[f]=down	∀f . bottom≤f ≤top Light[f]=down

∧ NC(Req,loc)

macros: !ReqAbove! = $f . loc <f £ top Ù Req[f]
!ReqBelow! = $f . Bottom £ f <loc Ù Req[f]

macro *NoChange:*
(Req’=Req) Ù (loc’=loc)

Inspection Method

NewDirection (dir, loc, Req, Light)

R_1 = (bottom ≤ loc ≤ top) ⇒

	!ReqAbove(loc)! = !ReqBelow(loc)!	¬ !ReqAbove(loc)! ∧ !ReqBelow(loc)!	!ReqAbove(loc)! ∧ ¬ !ReqBelow(loc)!
dir' =	'dir	down	up
Light'[] |	∀f . bottom ≤ f ≤ top Light'[f] = `dir	∀f . bottom ≤ f ≤ top Light'[f] = down	∀f . bottom ≤ f ≤ top Light'[f] = down

∧ NC(Req,loc)

```
Procedure NewDirection (var direction:enum; var Light:Vector; floor:integer) ;
var I : integer;
begin
   if  PendingAbove(floor) <> PendingBelow(floor) then begin
        if direction = up
           then direction := down
           else direction := up;
       for i := bottom to top do
           Light[i] := direction
   end
end;
```

PendingAbove(floor) External variables: Req

	∃f . [floor < f ≤ top ∧ Req[f]] =	
	true	false
result' =	true	false

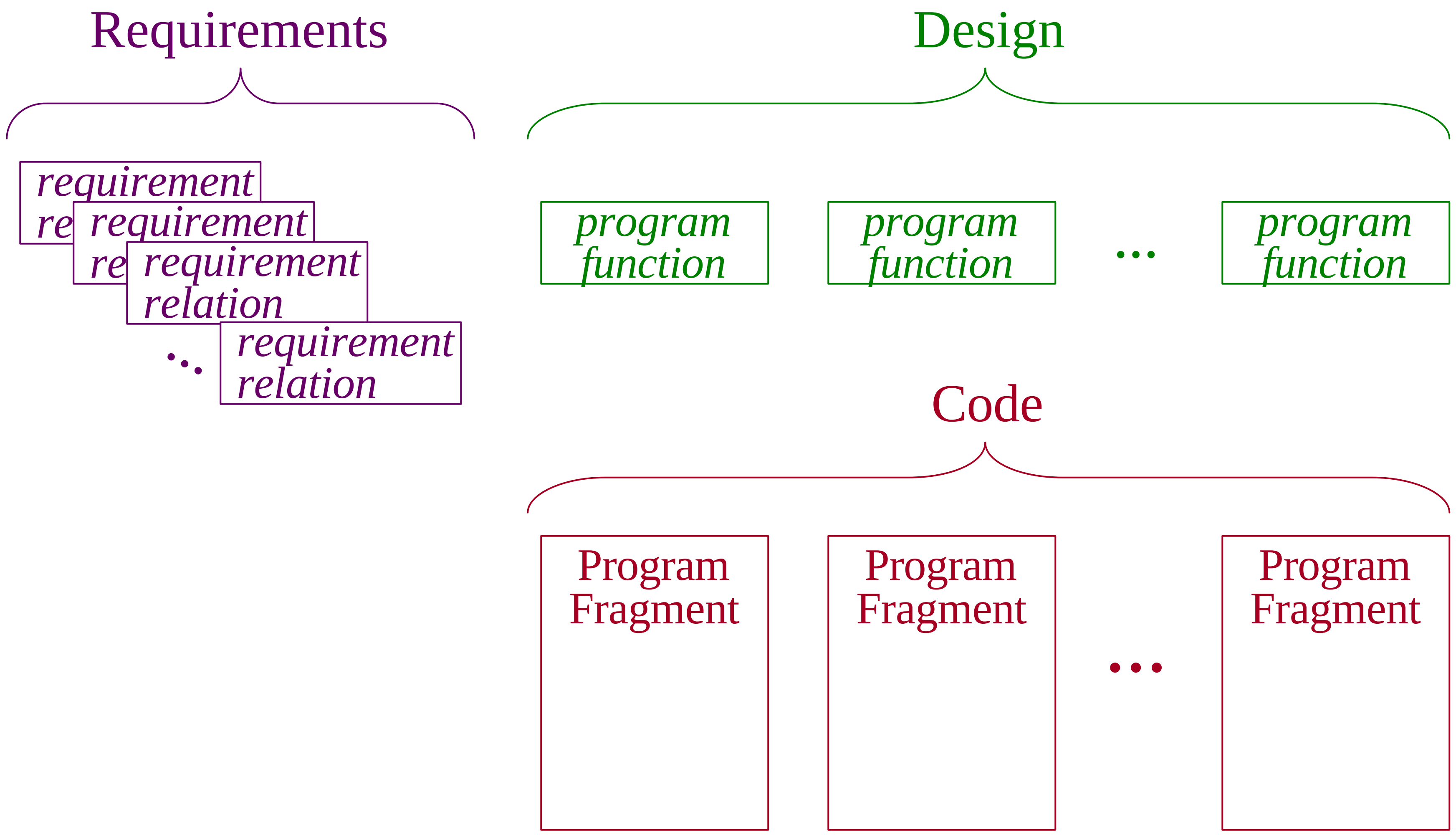
∧ NC(Req)

PendingBelow(floor) External variables: Req

	∃f . [bottom ≤ f < floor ∧ Req[f]] =	
	true	false
result' =	true	false

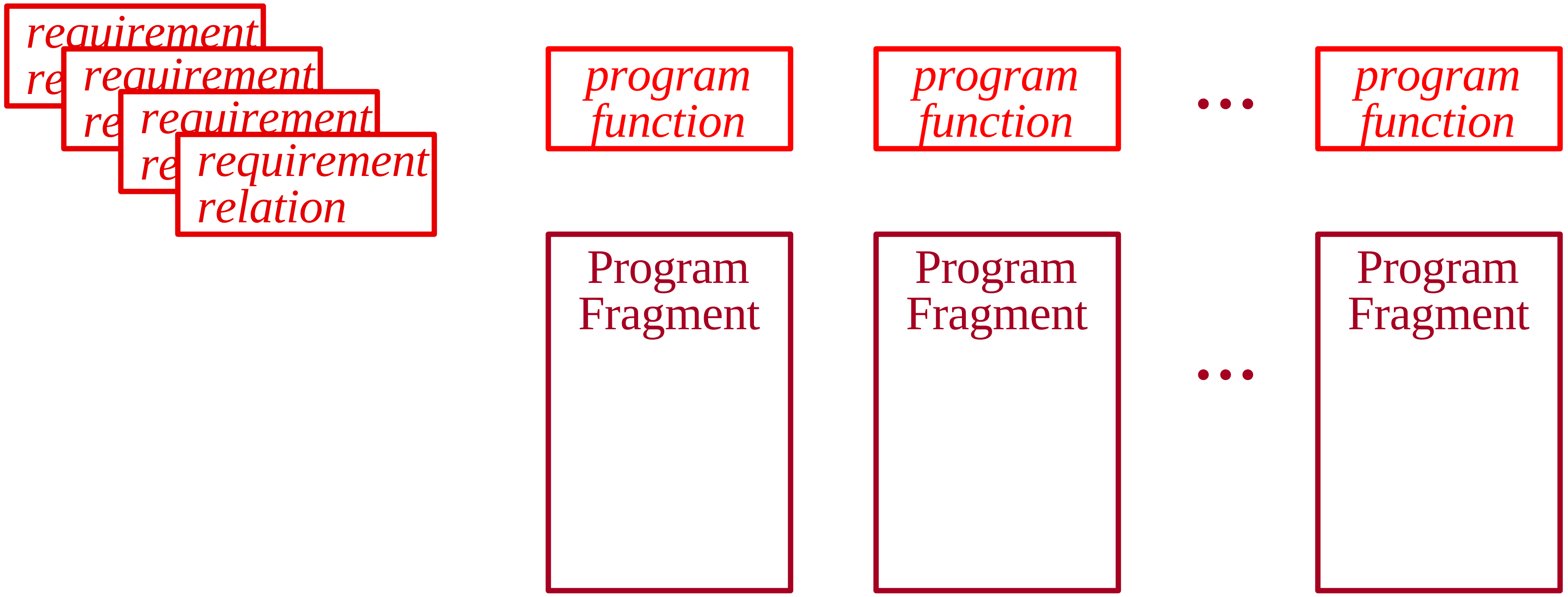
∧ NC(Req)

Systematic Inspections



Reviews and Inspections

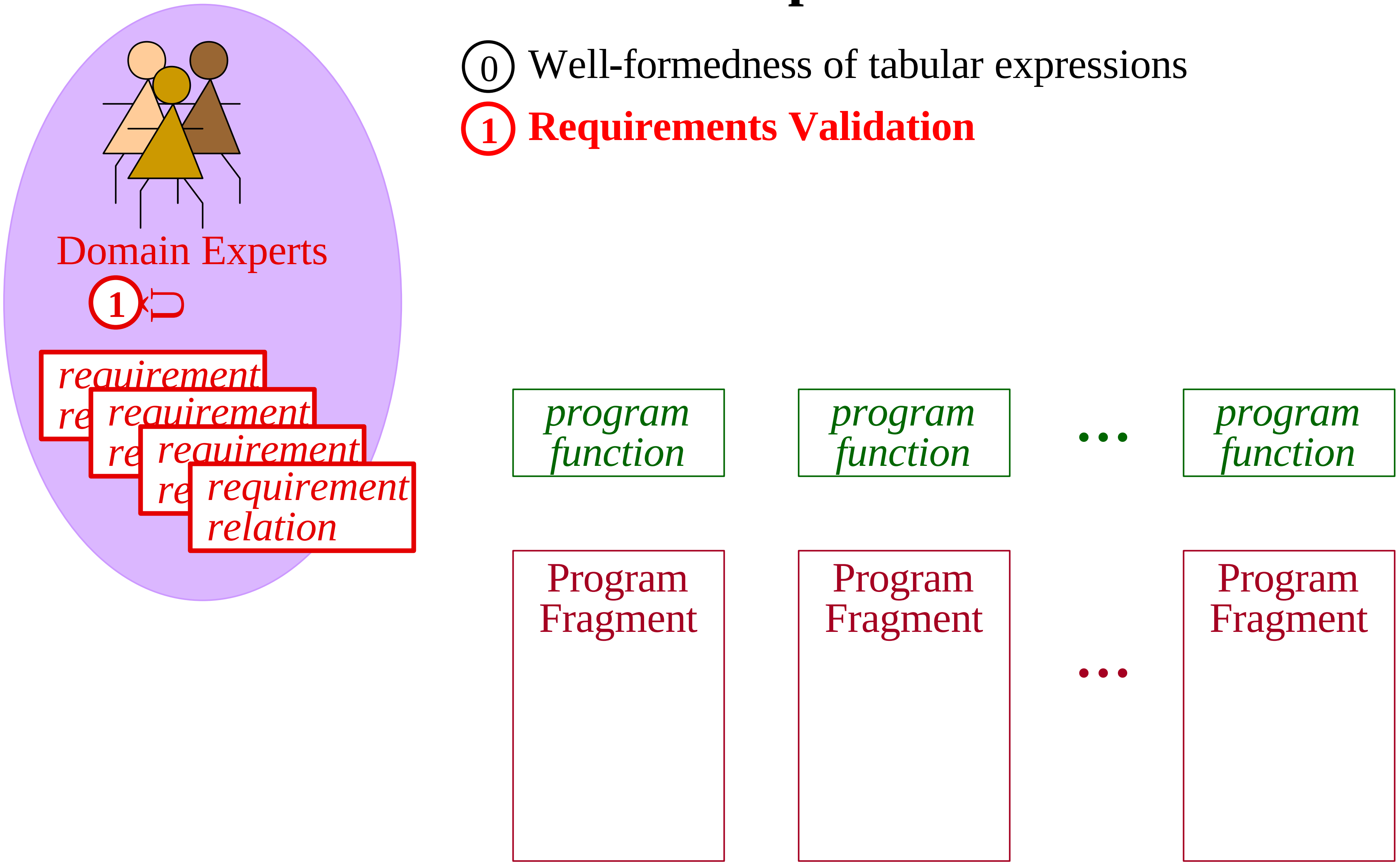
$\textcircled{0}$ Well-formedness of tabular expressions



Reviews and Inspections

(0) Well-formedness of tabular expressions

(1) Requirements Validation



Requirements Validation

Check that each case (subfunction, subrelation) produces the **correct output**.

Up	Down	Down	Up

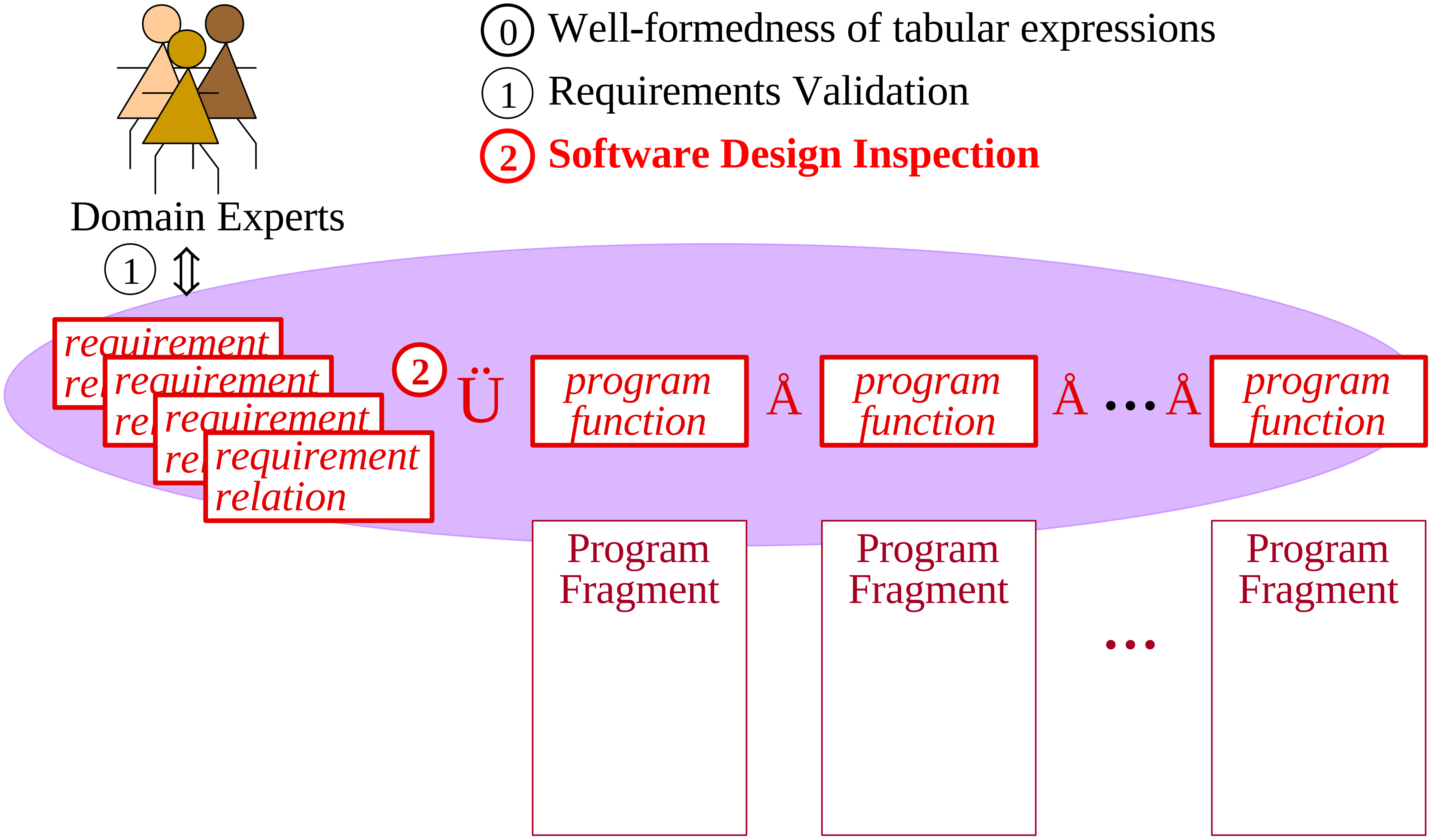
dir	Up	Up	Down	Down
$ f.(f³loc Ù Req[f])	true	false	---	true
$ f.(f≤loc Ù Req[f])	---	true	true	false

Reviews and Inspections

⓪ Well-formedness of tabular expressions

① Requirements Validation

② **Software Design Inspection**



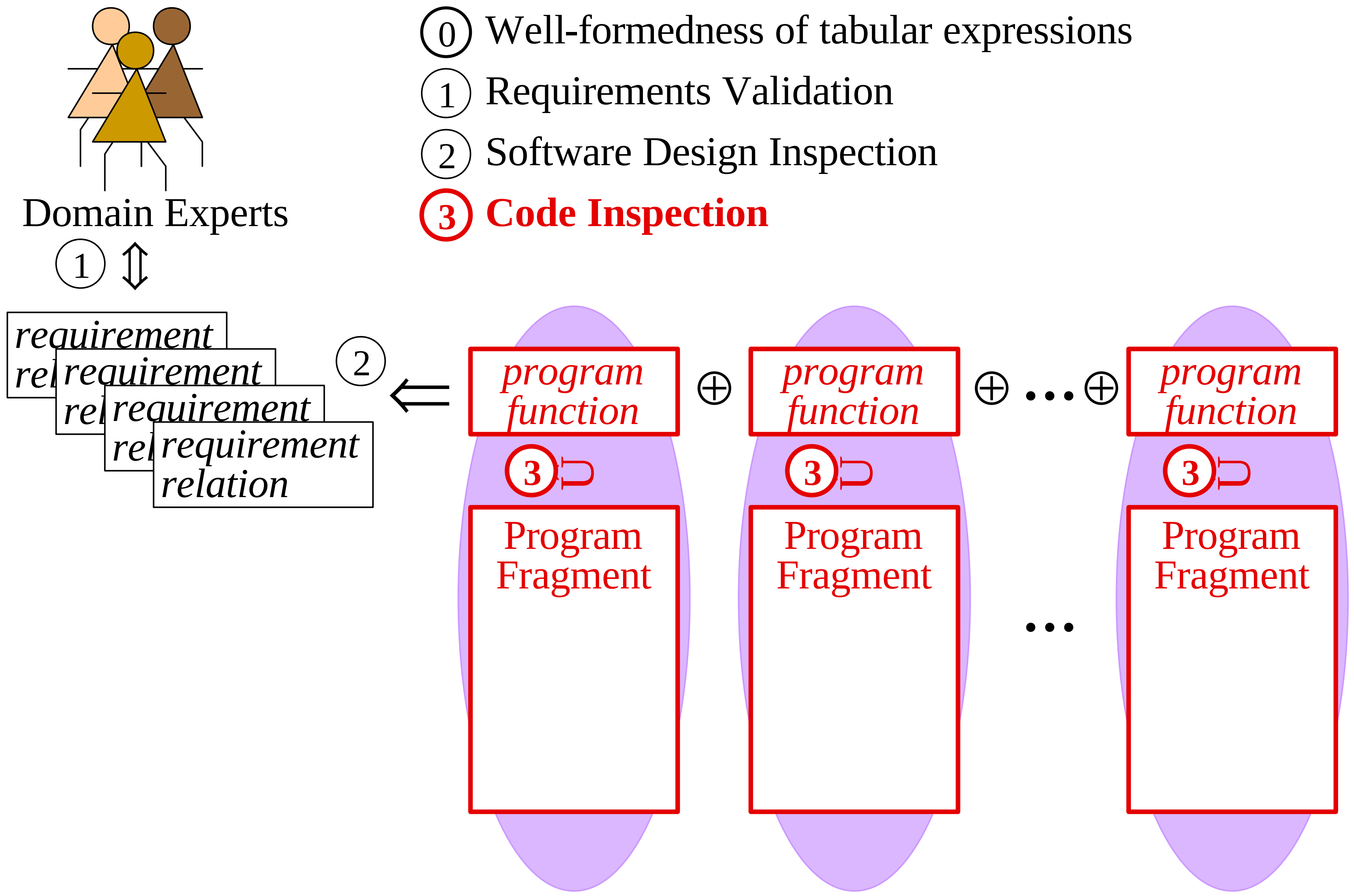
Reviews and Inspections

⓪ Well-formedness of tabular expressions

① Requirements Validation

② Software Design Inspection

③ Code Inspection



Darlington Experience

Since then:

Ontario Hydro and the Atomic Energy Canada Limited (AECL) have developed a family of standards, procedures, and guidelines for developing safety critical software for use in nuclear power plants, incorporating

- **tabular, mathematical representations** of requirements, design, and code
- **systematic inspections** of requirements
- **mathematical verification** or **rigorous argument** that
  - the design meets the requirements
  - the code meets the design

Other Experiences

Experiences in which practitioners adopted the technology

- A-7E, A-7D aircraft (SCR)
- Ontario Hydro nuclear plant applications (Parnas Tables)
- Lockheed C130-J transport aircraft (CoRE)
- Medtronic medical applications (SCR)

Experiences that involved practitioners

- Trident (submarine) Emergency Preset System (SCR)
- AT&T Service Evaluation System (Parnas Tables)
- Traffic and Collision Avoidance System (RSML)
- Aircraft Separation Minima (HOL Parnas Tables)
- International Space Station (SCR)

Formal Semantics of Tables

Several Table types look alike, and readers may **misinterpret** a Table’s meaning when they are given only the Table’s informal, **ad hoc semantics**.

	Up	Down	Down	Up
$ f.(f³loc Ù Req[f])	true	false	---	true
$ f.(f£loc Ù Req[f])	---	true	true	false
dir	Up	Up	Down	Down

Decision Table **OR** Inverted Table

Formal Semantics of Tables

To address this problem, there has been work on how to formulate the formal semantics of a tabular expression:

- **predicate rule p_T** to define the expression’s **domain**
- **relation rule r_T** to defines the expression’s **range**
- **composition rule C_T** to define how to combine subexpressions

	Up	Down	Down	Up
$ f.(f³loc Ù Req[f])	true	false	---	true
$ f.(f£loc Ù Req[f])	---	true	true	false
dir	Up	Up	Down	Down

Decision Table

$p_{T:}$ $H_2 = G$

r_T: H_3

C_T: $\cup_{j=1}^{3} (\text{Ä}_{i=1}^{4}\ F_{i,j})$

Inverted Table

$p_{T:}$ $H_2 \text{Ù} G$

r_T: H_3

C_T: $\cup_{i=1}^{4} (\cup_{j=1}^{3}\ F_{i,j})$

Table Transformations

One may want to **transform** one table to another representation, to formulate a more **compact expression** or **determine the equivalence** of two table expressions.

	Up	Down
dir=Up	$ f.(f³loc Ù Req[f])	Ø $ f.(f³loc Ù Req[f]) ∧ $ f.(f<loc Ù Req[f])
dir=Down	Ø $ f.(f£loc Ù Req[f]) Ù $ f.(f>loc Ù Req[f])	$ f.(f£loc Ù Req[f])

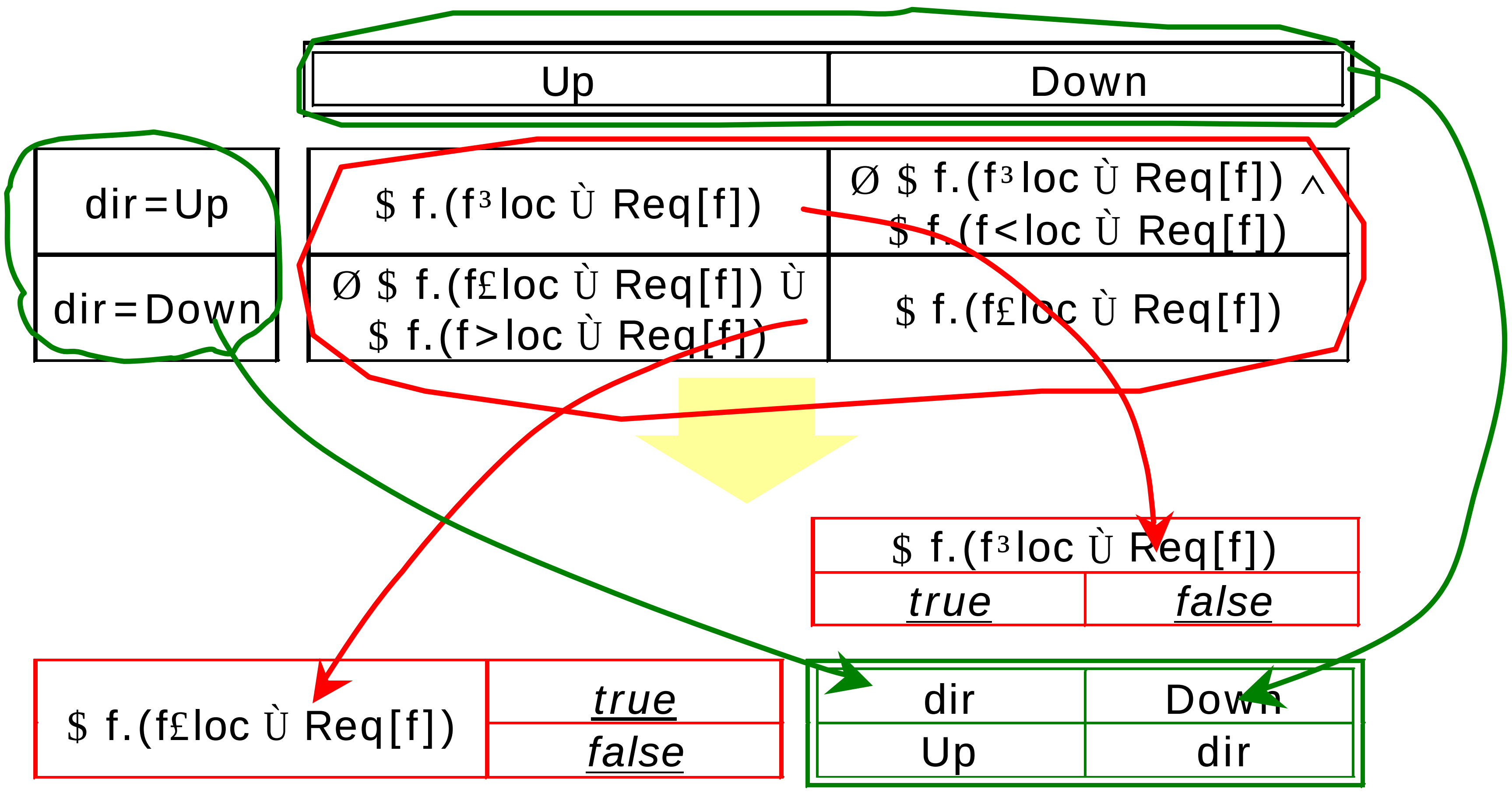
$ f.(f³loc Ù Req[f])	
true	*false*

$ f.(f£loc Ù Req[f])	*true*	dir | Down
	false	Up | dir

	dir	Down
	Up	dir

Table Transformations

But even a simple transformations, like one that **exchanges grid elements with header elements**, can require reorganization and simplification to produce a concise table.



	Up	Down
dir=Up	$ f.(f³loc Ù Req[f])	Ø $ f.(f³loc Ù Req[f]) ^ $ f.(f<loc Ù Req[f])
dir=Down	Ø $ f.(f£loc Ù Req[f]) Ù $ f.(f>loc Ù Req[f])	$ f.(f£loc Ù Req[f])

$ f.(f³loc Ù Req[f])	
true	*false*

$ f.(f£loc Ù Req[f])	*true*
	false

dir	Down
Up	dir

Automated Checking

Significant human effort may be needed to check that a table is **consistent** and that it **covers** the expression’s domain. Since these checks are application-independent and can be expressed as **constraints on predicates**, many can be automated.

	$Req[loc]$	$\neg Req[loc]$			
		$\neg\exists f.Req[f]$	$\exists f.(f>loc \wedge Req[f]) \wedge \neg\exists f.(f<loc \wedge Req[f])$	$\exists f.(f<loc \wedge Req[f]) \wedge \neg\exists f.(f>loc \wedge Req[f])$	$\exists f.(f<loc \wedge Req[f]) \wedge \exists f.(f>loc \wedge Req[f])$
dir’	dir	dir	Up	Down	dir
speed’	idle	idle	moving	moving	moving

Reasoning about Table Composition

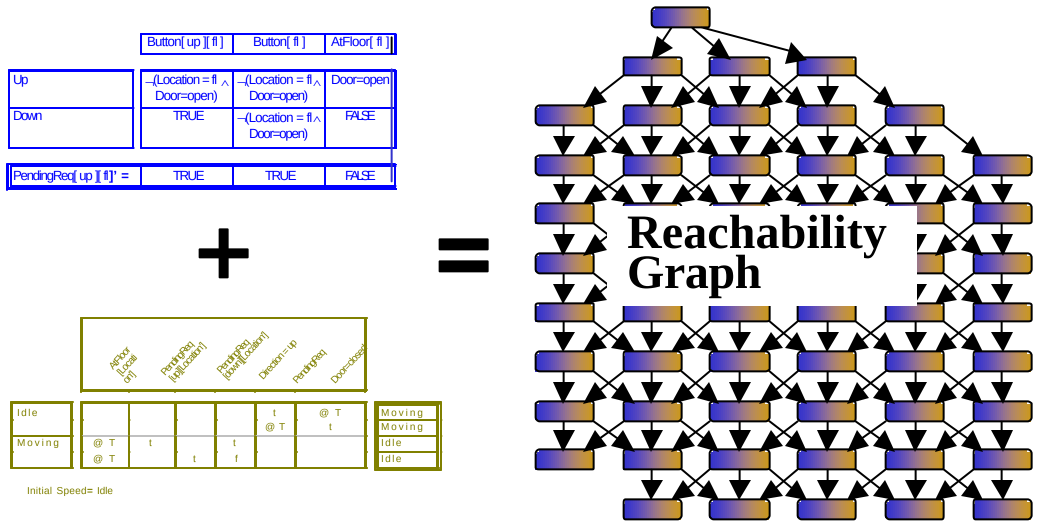
Each Table documents a separate concern.   If the concerns are **not completely separate** (e.g., if they react to changes in the same variables) then, we need to **review their composition**.

Application-Independent

- reachability
- deadlock
- cycle detection

Application-Dependent

- abstractions
- coordination
- safety properties
- liveness properties
- invariant generation



	Button[up][fl]	Button[fl]	AtFloor[fl]
Up	¬(Location = fl ∧ Door=open)	¬(Location = fl ∧ Door=open)	Door=open
Down	TRUE	¬(Location = fl ∧ Door=open)	FALSE
PendingReq[up][fl]' =	TRUE	TRUE	FALSE

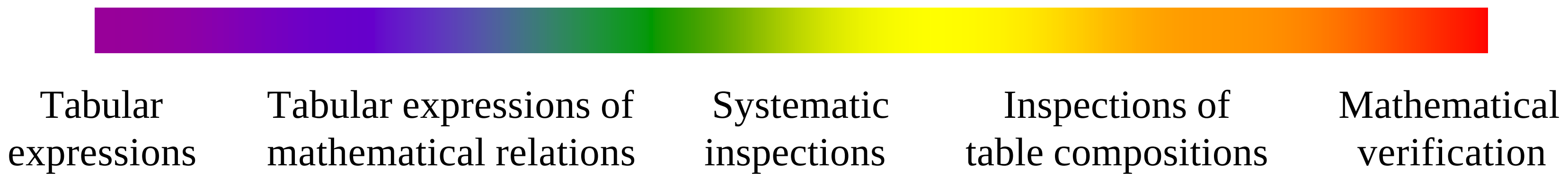
	AtFloor [Location']	PendingReq [up][Location']	PendingReq [down][Location']	Direction = up	PendingReq	Door=closed	
Idle					t	@ T	Moving
					@ T	t	Moving
Moving	@ T	t		t			Idle
	@ T		t	f			Idle

Initial Speed= Idle

Summary

Parnas Tables are **practical formalisms** that

- emphasize **abstraction** and **separation of concerns**
- are amenable to **readable**, **write-able**, and **review-able** yet precise software documents
- are **useful at different degrees of formalism**


Tabular expressions
Tabular expressions of mathematical relations
Systematic inspections
Inspections of table compositions
Mathematical verification