

# Feasibility in Event-B

Stefan Hallerstede

Department of Computer Science  
ETH Zurich  
Switzerland

Dagstuhl Seminar 06191 05/2006  
Rigorous Methods for Software Construction and Analysis

*Example “Identité de Bézout”  
contributed by Christophe Métayer, ClearSy, France*

This research is being carried out as part of the EU funded research project: IST 511599 RODIN (Rigorous Open Development Environment for Complex Systems).

- 1 Event-B
- 2 An Initial Model of Bézout's Identity
- 3 Feasibility
- 4 Development of the Algorithm
  - First Refinement
  - Second Refinement
  - More Refinements
- 5 Proof Statistics
- 6 Conclusion

- Event-B **models** are composed of **machines** and **contexts**
- *Contexts* provide **constants**, and **axioms** that describe properties of the constants.
- *Machines provide* **variables**, and **invariants** that describe properties of the variables, and **events** that describe possible changes of variable values.
- *Contexts* contain **static** parts of a model.
- *Machines* contain **dynamic** parts of a model.
- In this presentation we often simply speak of models instead of machine with context.

- An event has one of the following forms:
  - **begin**  $act(v)$  **end**
  - **when**  $grd(v)$  **then**  $act(v)$  **end**
  - **any**  $t$  **where**  $grd(t, v)$  **then**  $act(t, v)$  **end**

where  $v$  are the **variables** of the machine,  
 $act$  denotes the **action** of the event,  
 $grd$  denotes the **guard** of the event, and  
 $t$  are the **local variables** of the event.

- **begin**  $act(v)$  **end** = **when**  $\top$  **then**  $act(v)$  **end**

- Actions are modelled by simultaneous **assignments**.
- The most general form of assignment is the *non-deterministic assignment*.

Assignment	Before-After Predicate
$x :  prd(t, v, x')$	$prd(t, v, x') \wedge y' = y$

- The other assignments can be defined in terms of it:

*deterministic*     $x := exp(t, v)$      $\hat{=}$      $x :| x' = exp(t, v)$

*empty*     $skip$      $\hat{=}$      $v :| v' = v$

*member*     $x \in set$      $\hat{=}$      $x :| x' \in set$

- The before-after predicate of an action is the conjunction of its constituent assignments.

- *We do not present a behavioural semantics, but characterise the semantics by means of the proof obligations.*
- *Occurrences of events in formulas should be read as before-after predicates.*
- For all events  $evt$  we prove **feasibility**:

**evt/FIS**

$$ctx \wedge inv \wedge grd \Rightarrow \exists v' \cdot act$$

- By proving **evt/FIS** for each event  $evt$  we achieve that the syntactic guard  $grd$  correspond to the guard  $gd.evt$  of the event, i.e.

$$gd.evt = grd .$$

- In *initial models* we prove that **events preserve the invariant**:

**evt/INV**

$$ctx \wedge inv \wedge grd \wedge act \Rightarrow inv'$$

- In *refined models* we prove that the **abstract event**  $eabs$  **can simulate the refined event**  $eref$  :

**evt/SIM**

$$ctx \wedge inv \wedge inw \wedge gref \wedge aref \Rightarrow gabs \wedge (\exists v' \cdot aabs \wedge inw')$$

where  $gref$  and  $gact$  are the guards and action of the refined event, and  $gabs$  and  $aabs$  those of the abstract event.

- **New events** must be shown to **refine skip**:

**evt/NEW**

$$ctx \wedge inv \wedge inw \wedge gref \wedge aref \Rightarrow (\exists v' \cdot v' = v \wedge inw')$$

- **New events** must be shown to **to decrease a variant**:

**evt/VRT**

$$ctx \wedge inv \wedge inw \wedge gref \wedge aref \Rightarrow vrt \in \mathbb{N} \wedge vrt' < vrt$$

# Some Introductory Comments on the Development

- Using the feasibility proof obligation generated for events we could prove feasibility in the initial model.
- Instead, we prove feasibility in some arbitrary refinement step:

$$mch_1 \sqsubseteq mch_2 \sqsubseteq \dots \sqsubseteq mch_i \sqsubseteq mch_{i+1} \sqsubseteq \dots \sqsubseteq mch_n$$

where the red  $\sqsubseteq$  denotes a refinement step in which we prove feasibility.

- We present our approach to feasibility by way of an example:  
*Bézout's identity*.

# Bézout's Identity

For any two integers  $a$  and  $b$ , with  $a \neq 0$  and  $b \neq 0$ , there exist two integers  $u$  and  $v$  such that:

$$a * x + b * y \in \gcd[\{a \mapsto b\}] \quad .$$

## Remark.

This claim holds for any **principal ideal domain** (PID) instead of the integers. (This is what Bézout, 1730 - 1783, proved.)

For the algorithm to work that we are going to present the domain must be **Euclidean** (Euclidean implies PID). We stick to integers for simplicity.

# Intermezzo on *gcd* and *divides*

- **definition**

$$\textit{divides} \hat{=} \{x \mapsto y \mid x \neq 0 \wedge (\exists m \cdot m \in \mathbb{Z} \wedge y = x * m)\}$$

- **reflexivity**

$$\text{id}(\mathbb{Z} - \{0\}) \subseteq \textit{divides}$$

- **addition**

$$\forall x, y, z \cdot$$

$$x \mapsto y \in \textit{divides} \wedge x \mapsto z \in \textit{divides}$$

$$\Rightarrow$$

$$x \mapsto (y + z) \in \textit{divides}$$

- **scalar\_multiplication**

$$\forall x, y, z \cdot$$

$$x \mapsto y \in \textit{divides}$$

$$\Rightarrow$$

$$x \mapsto (y * z) \in \textit{divides}$$

- **definition**

$gcd \hat{=}$

$\{ (x \mapsto y) \mapsto d \mid$

$(\forall z \cdot z \mapsto x \in divides \wedge z \mapsto y \in divides \Rightarrow z \mapsto d \in divides) \}$

- **gcd\_not\_unique**

$\forall x, y \cdot x \mapsto y \in divides \wedge y \mapsto x \in divides \Rightarrow x = y \vee x = -y$

Using the following theorem we can solve Bézout's identity by means of relation *divides*:

## **gcd\_elimination**

$\forall d, u, v \cdot$

$$d = a * u + b * v \wedge$$

$$d \mapsto a \in \textit{divides} \wedge d \mapsto b \in \textit{divides}$$

$\Rightarrow$

$$d \in \textit{gcd}[\{a \mapsto b\}]$$

# Extended Euclidean Algorithm – Specification 1

## Initial Model

**constants**  $a, b$       **axm**  $a \in \mathbb{Z} \setminus \{0\} \wedge b \in \mathbb{Z} \setminus \{0\}$

**variables**  $d, u, v$       **inv0\_0**  $d \in \mathbb{Z} \wedge u \in \mathbb{Z} \wedge v \in \mathbb{Z}$

*init*  $\hat{=}$  **begin**

$d : \in \mathbb{Z}$

$u : \in \mathbb{Z}$

$v : \in \mathbb{Z}$

**end**

*bezout*  $\hat{=}$  **begin**

$d, u, v : |$

$d' = a * u' + b * v' \wedge$

$d' \mapsto a \in \text{divides} \wedge$

$d' \mapsto b \in \text{divides}$

**end**

# Feasibility of the Algorithm

- Feasibility of the sequential algorithm is defined by:

**FIS**

$$ctx \Rightarrow (\exists d', u', v' \cdot \mathit{init}; \mathit{bezout})$$

where “;” denotes sequential composition.

- We can prove this by means of the proof obligations of the model:
  - **init/FIS** – Feasibility of initialisation
  - **init/INV** – Invariant establishment
  - **bezout/FIS** – Feasibility of event *bezout*
- Using **gcd\_elimination**, Bézout’s identity follows from **FIS**.

# Feasibility of event *bezout*

- Feasibility of event *bezout*:

## **bezout/FIS**

$ctx \Rightarrow (\exists d', u', v' \cdot$

$$d' = a * u' + b * v' \wedge$$

$$d' \mapsto a \in \text{divides} \wedge d' \mapsto b \in \text{divides})$$

- How do we instantiate  $d', u', v'$  ?
- Unless we have a theory on prime ideal rings readily available, proving this claim requires some effort and is not obvious.

# Extended Euclidean Algorithm – Implementation

- Our aim is to construct an algorithm.
- This algorithm computes witnesses  $d, u, v$  for **FIS**:
- |  |  |
|--|--|
| $s, t, q, r := \emptyset, \emptyset, \emptyset, \emptyset;$<br>$tp := 0;$<br>$s(0) := a;$<br>$t(0) := b;$<br>$q(0) := a \div b;$<br>$r(0) := a \bmod b;$ | <b>while</b> $r(tp) \neq 0$ <b>do</b><br>$s(tp+1) := t(tp)$<br>$t(tp+1) := r(tp)$<br>$q(tp+1) := t(tp) \div r(tp)$<br>$r(tp+1) := t(tp) \bmod r(tp)$<br><b>end</b><br>$di, ui, vi := t(tp), 1, 1 - q(tp);$<br><b>while</b> $tp \neq 0$ <b>end</b><br>$ui, vi, tp := vi, ui - q(tp-1)*vi, tp-1$<br><b>end;</b><br>$d, u, v := di, ui, vi$ |
|--|--|
- By refinement we prove that the result  $d, u, v$  of the computation provides suitable witnesses.

- At some point feasibility of the specification is shown; otherwise there would not be an algorithm.
- The algorithm provides the witnesses whenever the computation does not get blocked and terminates.
- Hence, relying on these two properties we should be able to prove feasibility **FIS**.

# Proving Feasibility

## Simplified Case

- Assume, we do not introduce any new event during refinement.
- We want to show **FIS**:  $ctx \Rightarrow (\exists v' \cdot iabs ; eabs)$  .
- In each refinement step we prove that the guard of the event is strengthened (by **evt/SIM**)

$$ctx \wedge inv \wedge inw \Rightarrow (gd.eref \Rightarrow gd.eabs)$$

where  $eref$  is the refined event and  $eabs$  is the abstract event.

- If we prove

$$ctx \wedge inv \wedge inw \Rightarrow gd.eref$$

in the refined model, then  $ctx \wedge inv \wedge inw \Rightarrow gd.eabs$  follows.

- The extra assumption  $inv \wedge inw$  is established by initialisation (using **init/FIS** and **init/SIM**).
- We have  $ctx \Rightarrow (\exists v', w'. iabs \wedge inv' \wedge inw')$ , so we can infer

$$ctx \Rightarrow (\exists v'. iabs \wedge gd.eabs')$$

which is just a variant of  $ctx \Rightarrow (\exists v'. iabs; eabs)$ , i.e. **FIS**.

# Proof Obligation for Feasibility

## General Case

- Let  $EREF$  be the events of a refined model, and  $eabs$  be an abstraction of  $eref \in EREF$ . Then it is sufficient to prove that

### DLF

$$ctx \wedge inv \wedge inw \Rightarrow (\exists e \in EREF \cdot gd.e) ,$$

to show feasibility **FIS** of the abstract model.

- The proof of this claim uses that

$$(EREF - \{eref\})^*; eref \text{ refines } eabs$$

where  $(EREF - \{eref\})^*$  denotes iteration, and the fact that the iteration terminates.

# Extended Euclidean Algorithm – Specification 2

- We change the appearance of event *bezout*:

*bezout*  $\hat{=}$  **any** *dd, uu, vv* **where**  
*dd* = *a* \* *uu* + *b* \* *vv*  
*dd*  $\mapsto$  *a*  $\in$  *divides*  
*dd*  $\mapsto$  *b*  $\in$  *divides*

**then**

*d, u, v* := *dd, uu, vv*

**end**

- For this model, proof obligation **bezout**/**FIS** is trivially true.
- But **FIS** can no longer be proved initially.
- Also note the guard of event *bezout*,

$\exists dd, uu, vv \cdot$

*dd* = *a* \* *uu* + *b* \* *vv*  $\wedge$

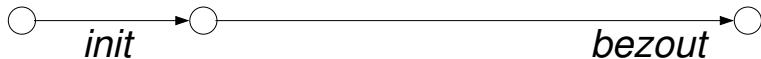
*dd*  $\mapsto$  *a*  $\in$  *divides*  $\wedge$  *dd*  $\mapsto$  *b*  $\in$  *divides* .

# Development Overview

- We introduce the algorithm in **two refinement steps**.
- This corresponds to the **two phases** (i.e. two loops) of the algorithm.
  - **up-phase**: compute a stack of divisions.
  - **down-phase**: compute the coefficients  $u$  and  $v$ .
- We prove **DLF**, hence, **feasibility** in the second refinement.
- We carry out two more refinements afterwards so that we can use **event composition rules for sequential programs**.
- The last two refinements themselves are not very interesting; but it is important that the **we can continue to refine the model after having proved feasibility**.

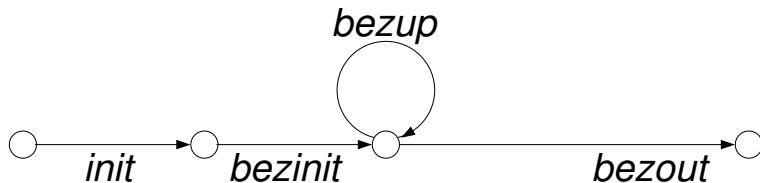
# Development Overview

## Initial Model



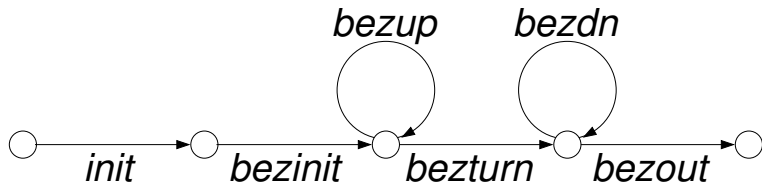
# Development Overview

## First Refinement



# Development Overview

## second Refinement



# First Refinement

## Example of Stack of Divisions

- Start with  $a = 448$  and  $b = 92$ :

$$\begin{array}{rclclcl} tx & : & s & = & t & * & q & + & r \\ 0 & : & 448 & = & 92 & * & 4 & + & 80 \\ 1 & : & 92 & = & 80 & * & 1 & + & 12 \\ 2 & : & 80 & = & 12 & * & 6 & + & 8 \\ 3 & : & 12 & = & 8 & * & 1 & + & 4 \\ 4 & : & 8 & = & 4 & * & 2 & + & 0 \end{array}$$

- The last remainder  $r(4)$  is zero.
- The remainder is a good candidate for a variant.
- The last divisor  $t(4)$  is a gcd of  $a$  and  $b$ .

# First Refinement

## Properties of the Stack – Invariant 0

• **variables**  $tx, s, t, q, r, up$

• **inv1\_0**

$up \in \text{BOOL} \wedge$  describes whether the **up-phase** of the algorithm has been started.

$tx \in \mathbb{N} \wedge$  denotes the **top of the stack**.

$s \in \mathbb{N} \mapsto \mathbb{Z} \wedge$  denotes the stack of **dividends**.

$t \in \mathbb{N} \mapsto \mathbb{Z} \wedge$  denotes the stack of **divisors**.

$q \in \mathbb{N} \mapsto \mathbb{Z} \wedge$  denotes the stack of **quotients**.

$r \in \mathbb{N} \mapsto \mathbb{Z}$  denotes the stack of **remainders**.

# First Refinement

## Properties of the Stack – Invariant 1

- **inv1\_1**

$up = \text{TRUE} \Rightarrow$

- The stack is well formed:

$$s \in 0 .. tx \rightarrow \mathbb{Z} \wedge$$

$$t \in 0 .. tx \rightarrow \mathbb{Z} \wedge$$

$$q \in 0 .. tx \rightarrow \mathbb{Z} \wedge$$

$$r \in 0 .. tx \rightarrow \mathbb{Z} \wedge$$

- The bottom of the stack contains the two parameters:

$$s(0) = a \wedge t(0) = b \wedge$$

- Each layer on the stack represents a division:

$$\forall x \cdot x \in 0 .. tx \Rightarrow s(x) = t(x) * q(x) + r(x) \wedge$$

- All divisors are not zero:

$$\forall x \cdot x \in 0 .. tx \Rightarrow t(x) \neq 0 \wedge$$

- Consecutive layers on the stack are connected:

$$\forall x \cdot x \in 1 .. tx \Rightarrow t(x-1) = s(x) \wedge$$

$$\forall x \cdot x \in 1 .. tx \Rightarrow r(x-1) = t(x)$$

# First Refinement

## Refined Event *bezout*

- The guard of event *bezout* is strengthened:

*bezout*  $\hat{=}$  **any** *dd, uu, vv* **where**  
 $dd = a * uu + b * vv$   
 $dd \mapsto a \in \text{divides}$   
 $dd \mapsto b \in \text{divides}$   
*up* = TRUE  
 $r(tx) = 0$   
**then**  
 $d, u, v := dd, uu, vv$   
**end**

- Event *bezout* only imposes the additional fact that the *up*-phase has terminated:  $up = \text{TRUE} \wedge r(tx) = 0$ .
- Two new events *bezinit* and *bezup* are introduced.

# First Refinement

New Events *bezinit* and *bezup*

*bezinit*  $\hat{=}$

**when**

$up = \text{FALSE}$

**then**

$up := \text{TRUE}$

$tx := 0$

$s := \{0 \mapsto a\}$

$t := \{0 \mapsto b\}$

$q := \{0 \mapsto a \div b\}$

$r := \{0 \mapsto a \bmod b\}$

**end**

*bezup*  $\hat{=}$

**when**

$up = \text{TRUE}$

$r(tp) \neq 0$

**then**

$tx := tp + 1$

$s(tx+1) := t(tx)$

$t(tx+1) := r(tx)$

$q(tx+1) := t(tx) \div r(tx)$

$r(tx+1) := t(tx) \bmod r(tx)$

**end**

- variant: **vrt1** ( $\{\text{TRUE} \mapsto \text{abs}(r(tp)), \text{FALSE} \mapsto \text{abs}(b)\})(up)$

# First Refinement

Intermezzo on *division* and *abs*

- **definition**

$$abs \hat{=} id(\mathbb{N}) \cup \{x \mapsto y \mid x < 0 \wedge y = -x\}$$

- **division**

$$\forall x, y \cdot m \neq 0 \wedge n \neq 0 \Rightarrow m = n * (m \div n) + (m \bmod n)$$

- **remainder**

$$\forall m, n \cdot m \neq 0 \wedge n \neq 0 \Rightarrow abs(m \bmod n) < abs(n)$$

- **Euclidean**

$$\forall m, n \cdot m \neq 0 \wedge n \neq 0 \Rightarrow (\exists x, y \cdot m = n * x + y \wedge abs(y) < abs(n))$$

- We can prove convergence by means of variant **vrt1** because **remainder** holds in **Euclidean** rings, in particular, in  $\mathbb{Z}$ .

# First Refinement

## Refined Initialisation

*init*  $\hat{=}$  **begin**

*d* :  $\in \mathbb{Z}$

*u* :  $\in \mathbb{Z}$

*v* :  $\in \mathbb{Z}$

*s, t, q, r* :=  $\emptyset, \emptyset, \emptyset, \emptyset$

*tx* :  $\in \mathbb{N}$

*up* := **FALSE**

**end**

- The *up*-phase has not started initially.

# Second Refinement

## Example of Stack of Divisions

- The stack of divisions and the derivation of coefficients:

$$\begin{array}{rclclclcl} tx & : & s & = & t & * & q & + & r \\ 0 & : & 448 & = & 92 & * & 4 & + & 80 \\ 1 & : & 92 & = & 80 & * & 1 & + & 12 \\ 2 & : & 80 & = & 12 & * & 6 & + & 8 \\ 3 & : & 12 & = & 8 & * & 1 & + & 4 \\ 4 & : & 8 & = & 4 & * & 2 & + & 0 \\ di & = & s & * & ui & + & t & * & vi \\ 4 & = & 8 & * & 1 & + & 4 & * & -1 \\ 4 & = & 12 & * & -1 & + & 8 & * & 2 \\ 4 & = & 80 & * & 2 & + & 12 & * & -13 \\ 4 & = & 92 & * & -13 & + & 80 & * & 15 \\ 4 & = & 448 & * & 15 & + & 92 & * & -73 \end{array}$$

- The number of iterations of the second phase is bound by the size of the stack

# Second Refinement

## Properties of the Coefficients – Invariant 0

- **variables**  $tx, s, t, q, r, up, i, di, ui, vi$

- **inv2\_0**

$dn \in \text{BOOL} \wedge$  describes whether the **dn-phase** of the algorithm has been started.

$i \in \mathbb{N} \wedge$  denotes the **position in the stack**.

$di \in \mathbb{Z} \wedge$  denotes the **gcd**.

$ui \in \mathbb{Z} \wedge$  denotes the **coefficient of a**.

$vi \in \mathbb{Z} \wedge$  denotes the **coefficient of b**.

# Second Refinement

## Properties of the Coefficients – Invariants 1 & 2

- **inv2\_1**

$dn = \text{TRUE} \Rightarrow$

- The current position lies within the stack boundaries:  
 $i \in 0 .. tx \wedge$
- The gcd is maintained during each iteration (compare to  $gd.bezout$ ):

$$di \mapsto s(i) \in \text{divides} \wedge$$

$$di \mapsto t(i) \in \text{divides} \wedge$$

$$di = s(i) * ui + t(i) * vi \wedge$$

- the last remainder on the stack is zero:

$$r(tx) = 0$$

- **inv2\_1**

The  $dn$ -phase takes place after the  $up$ -phase:

$$up = \text{FALSE} \Rightarrow dn = \text{FALSE} \wedge$$

$$dn = \text{TRUE} \Rightarrow up = \text{TRUE}$$

# Second Refinement

## Refined Event *bezout*

- The guard of event *bezout* is strengthened:

*bezout*  $\hat{=}$  **when**

*up* = TRUE

*dn* = TRUE

*i* = 0

**then**

*d, u, v* := *di, ui, vi*

**end**

- Event *bezout* now requires that the two phases have finished.
- It simply copies the result of the computation into the variables

*d, u, v* .

- Two new events *bezturn* and *bezdn* are introduced.

# Second Refinement

New Events *bezturn* and *bezdn*

*bezturn*  $\hat{=}$

**when**

$up = \text{TRUE}$

$dn = \text{FALSE}$

$r(tx) = 0$

**then**

$dn := \text{TRUE}$

$di := t(tx)$

$i := tx$

$ui := 1$

$vi := 1 - q(tx)$

**end**

*bezup*  $\hat{=}$

**when**

$dn = \text{TRUE}$

$i > 0$

**then**

$ui := vi$

$vi := ui - q(i-1) * vi$

$i := i-1$

**end**

- variant: **vrt2**  $(\{ \text{TRUE} \mapsto i, \text{FALSE} \mapsto tp+1 \})(dn)$

# Second Refinement

## Feasibility of the Model

- We have to show:

$$ctx \wedge inv \Rightarrow$$

$$gd.bezout \vee gd.bezinit \vee gd.bezup \vee gd.bezturn \vee gd.bezdn$$

that is:

$$ctx \wedge inv \Rightarrow$$

$$(up = \text{TRUE} \wedge dn = \text{TRUE} \wedge i = 0) \vee$$

$$(up = \text{FALSE}) \vee$$

$$(up = \text{TRUE} \wedge r(tx) \neq 0) \vee$$

$$(up = \text{TRUE} \wedge r(tx) = 0 \wedge dn = \text{FALSE}) \vee$$

$$(dn = \text{TRUE} \wedge i > 0)$$

- This is easy!

# Towards a Sequential Program

- We want to continue refining because:
  - The Event-B composition rules require the guards to be of a particular form.
  - Variable  $i$  is redundant.  
We can replace variables  $tx$  and  $i$  by one variable  $tp$  that represents always the top of the stack.
- *Problem:* we only have proved that guards are **strengthened**.
- If the new refinements ought to be feasible, too, we must prove that the guards are also **weakened**.

# Preserving Feasibility

## (Strong) Guard Weakening

- Guard Weakening Proof Obligation:

### **GWK**

$$ctx \wedge inv \wedge inw \Rightarrow \\ (gd.eabs \Rightarrow gd.eref \vee (\exists e \in ENEW(eref) \cdot gd.e))$$

where the events  $ENEW$  of the refinement are partitioned  $ENEW(eref)$  between the existing events  $eref$  (that refine abstract events  $eabs$ )

- We have proved these proof obligations for the first two refinements without mentioning it in this presentation.
- It is also needed for (sequential) event composition and, hence, does not introduce any extra effort.

- If we have proved

$$ctx \wedge inv \Rightarrow (\exists e \in EABS \cdot gd.e)$$

for a model,

- then by **GWK** we have in the refinement

$$ctx \wedge inv \wedge inw \Rightarrow (\exists e \in EREF \cdot gd.e)$$

hence, feasibility is preserved

- The method to prove feasibility along a sequence of refinements now looks like this:

$$gd.EVT_1 \Leftarrow \dots \Leftarrow gd.EVT_i \Rightarrow \dots \Rightarrow gd.EVT_n$$

# Feasibility of the Extended Euclidean Algorithm

(Strong) Guard Weakening

bezout0  
refines

in the first two refinements  
the algorithm is developed



bezout1  
realises



bezout2  
refines

in the second refinement  
feasibility is proved



bezout3  
refines

in the following refinements  
feasibility is preserved



bezout4

finally, the algorithm is implemented

# Development Proof Statistics

- Refinement Proofs:

Name	Proof Obligations	Automatic	Manual	% (Auto)
bezout0	1	0	1	0 %
bezout1	51	35	16	67 %
bezout2	28	20	8	71 %
bezout3	10	10	0	100 %
bezout4	15	15	0	100 %
$\Sigma$	105	80	25	76 %

- Mathematical theory (not all may be necessary):

Name	Proof Obligations	Automatic	Manual	% (Auto)
abs_thy	10	0	10	0 %
div_thy	3	0	3	0 %
divides_thy	4	1	3	25 %
gcd_thy	1	0	1	0 %

- We have shown a method to prove feasibility of sequential programs in Event-B that works well with refinement.
- There was only one additional proof obligation that was easy to discharge.
- We think the proof is always easy because:
  - Sequential algorithms usually contain very simple guards in loops and conditional statements.
  - These guards must appear at some point in the refinement.
  - In general, at some point in a refinement we usually achieve a model that contains guards with only very little nondeterminism.
- This provides more evidence that Event-B is also well-suited for sequential program development.