Refinement-based Guidelines for Constructing Algorithms

Dominique Méry
Université Henri Poincaré Nancy 1

Meeting at UDINE
IFIP WG 1.3
September 11-12, 2009

Summary

- ◇ Adding guidelines and hints for developing algorithms
- ◇ A framework for teaching programming methodology based on the famous pre/post specifications, together with the refinement.
- ◇ Illustrating a methodology based on Event B and the refinement by developing algorithms: insertion sorting,Floyd's algorithm, CYK algorithm, . . .
- ◇ Tools based on Event B supported by the RODIN platform
- ◇ Current works: cryptologic algorithms, access control systems, distributed algorithms, . . .

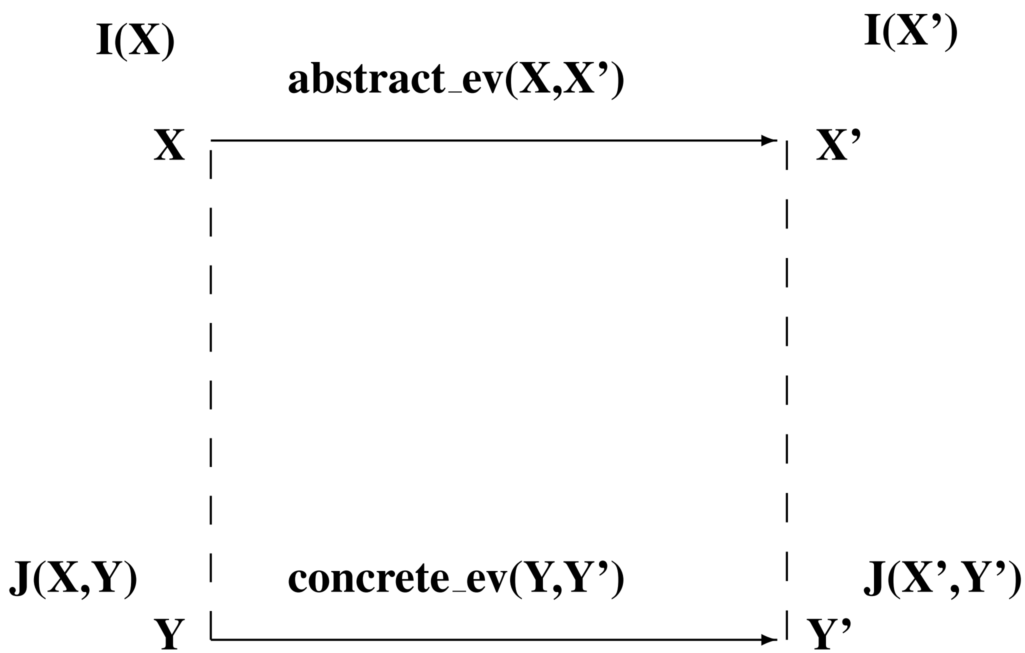
Papers

◇ D. Cansell, D. Méry. – -Proved-Patterns-Based Development for Structured Programs.-. – *In : Computer Science - Theory and Applications, Second International, Symposium on Computer Science in Russia - CSR 2007*, Volker Diekert, Mikhail V. Volkov, Andrei Voronkov (réd.), *Lecture Notes in Computer Science*, *4649*, Springer, pp. 104–114. – Ekaterinburg, Russia, 2007.

◇ D. Méry. – -A simple refinement-based method for constructing algorithms-. – *SIGCSE Bull. 41*, 2 (2009), pp. 51–59.

◇ D. Méry. – -Refinement-based guidelines for algorithmic systems-. – *International Journal of Software and Informatics* (2009), p. 35 pages. – to appear.

Refinement



Using formal method by guidelines application

- ◇ Defining a framework relating the world of algorithms and the world of (Event B) models

- ◇ Providing as much as possible mechanized steps

- ◇ Making proofs as simple as possible

- ◇ Helping to derive invariants from definitions

Proof-based development: Call as Event Guideline

$$
\begin{tikzcd}
\mathtt{CALL} \arrow[r, "call{-}as{-}event"] \arrow[d, "call"'] & \textbf{PREPOST} \arrow[r, "\textbf{SEES}"] \arrow[d, "\textbf{REFINEMENT}"'] & \textbf{PB} \arrow[d, "\textbf{EXTENDS}"'] \\
\mathtt{PROCEDURE} & \textbf{M} \arrow[l, "mapping"'] \arrow[r, "\textbf{SEES}"] & \textbf{CM}
\end{tikzcd}
$$

◇ `CALL` is the call of the `PROCEDURE`

◇ PREPOST is the machine containing the events stating the pre- and postconditions of `CALL` and `PROCEDURE`, and M is the refinement machine of PREPOST, with events including control points defined in CM.

◇ The *call-as-event* transformation produces a model PREPOST and a context PB from `CALL`.

◇ The *mapping* transformation allows us to derive an algorithmic procedure that can be mechanized.

◇ `PROCEDURE` is a node corresponding to a procedure derived from the refinement model M. `CALL` is an instantiation of `PROCEDURE` using parameters x and y.

◇ M is a refinement model of PREPOST, which is transformed into `PROCEDURE` by applying structuring rules. It may contain events corresponding to calls of other procedures.

Formal development of sequential algorithms

procedure $\texttt{PROC}(x;$ **var** $y)$
precondition $P(x)$
postcondition $Q(x,y)$

- ◇ Using the design-by-contract approach
- ◇ Progressive introduction of the methodology on non-trivial examples.
- ◇ Introduction of concepts of programming (call-by-value,...) and of modelling (constants, axioms, ...)
- ◇ Using diagrams to improve the communication with students through definitions (dynamic programming)
- ◇ Organizing the global interactions between modelling and proving.

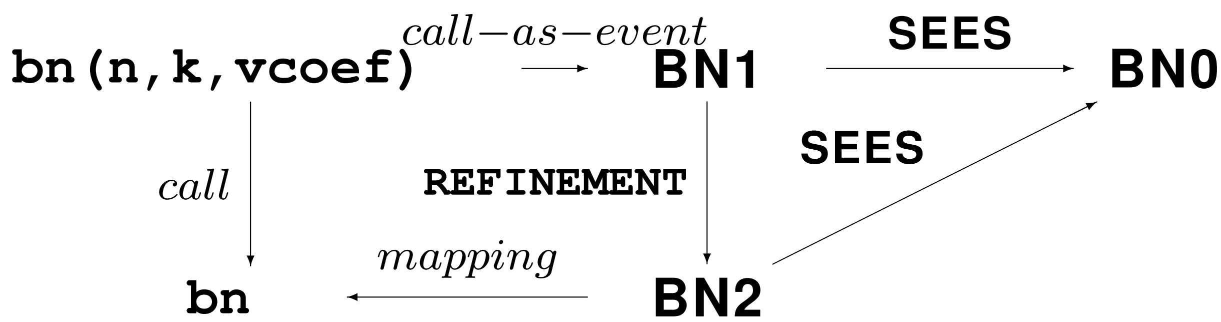
Three problems to solve

- ◇ **Presenting the method: Computing binomial coefficients**
- ◇ **Illustrating one classical example: Sorting by insertion**
- ◇ **Using the dynamic programming: Floyd’s algorithm**

Filling the Call as Event Guideline

$$
\begin{tikzcd}
\texttt{CALL} \arrow[r, "call-as-event"] \arrow[d, "call"'] & \textbf{?PREPOST} \arrow[r, "\textbf{SEES}"] \arrow[d, "\textbf{REFINEMENT}"'] & \textbf{?PB} \arrow[d, "\textbf{EXTENDS}"'] \\
\texttt{?PROCEDURE} & \textbf{?M} \arrow[l, "mapping"'] \arrow[r, "\textbf{SEES}"] & \textbf{?CM}
\end{tikzcd}
$$

Problem 1: Computing the binomial coefficients



◇ The computation of binomial coefficients is based on Pascal's triangle and we define it as a partial function c.

◇ Data n and k are defined in the context called $BN0$.

◇ The call $bn(n, k, vcoef)$ is translated as an event which is simply setting $vcoef$ to $c(n \mapsto k)$.

◇ The refinement $BN2$ produces a collection of events analysing the different steps of the computations required for computing the value of $c(n \mapsto k)$.

Comments on the application 1

- ◇ Pascal's triangle provides a graphical guide for writing d's definition into $BN0$.

- ◇ We have introduced another graphical structure for supporting the case analysis related to the values of n and k and the introduction of control flow.

- ◇ The world of mathematics is defining a value $c(n \mapsto k)$ and the world of computing will derive a process using c and its definition for producing the same value.

  ...

Comments on the application 1

- **The refinement i is guided by three cases for the call instances:**
  - **Either k is 0,**
  - **or k is n,**
  - **or is neither 0, nor n.**

- **Let us consider the difficult case: $k \neq 0$ and $k \neq n$:**

$$\forall k \in \{1, \ldots n-1\}. \binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \tag{1}$$

  - **Using the same event for computing $\binom{n-1}{k-1}$ and $\binom{n-1}{k}$.**
  - **These events are translated into a recursive call by the mapping.**
  - **the final computing event is computing the value of the sum of the two values.**

Comments on the application 1

◇

INVARIANTS

$inv1 : l \in LOC$

$inv3 : vtcoefx \in \mathbb{N}$

$inv4 : vtcoefy \in \mathbb{N}$

$inv5 : l \in \{callx, cally, endcalling\} \Rightarrow k \neq 0 \wedge n \neq 0 \wedge k < n$

$inv6 : l = cally \Rightarrow vtcoefx = c(n-1 \mapsto k-1)$

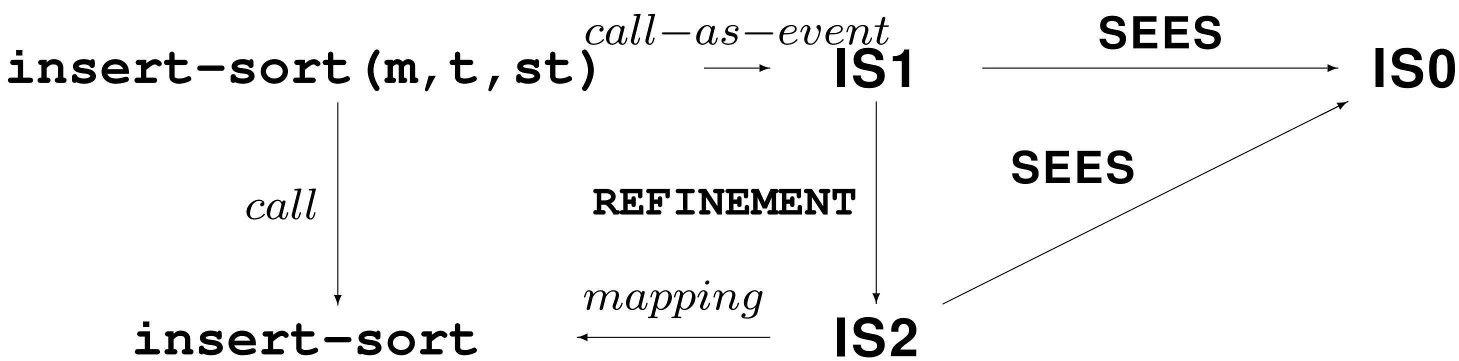
$inv7 : l = endcalling \Rightarrow vtcoefy = c(n-1 \mapsto k) \wedge vtcoefx = c(n-1 \mapsto k-1)$

$inv8 : l = end \Rightarrow vcoef = c(n \mapsto k)$

◇ **The refinement produces 42 proof obligations and 2 were manual. The other proof obligations are automatically discharged.**

Note: The example is simple and the function c is easy to define. The invariant is built by analysing the expression of the computed value.

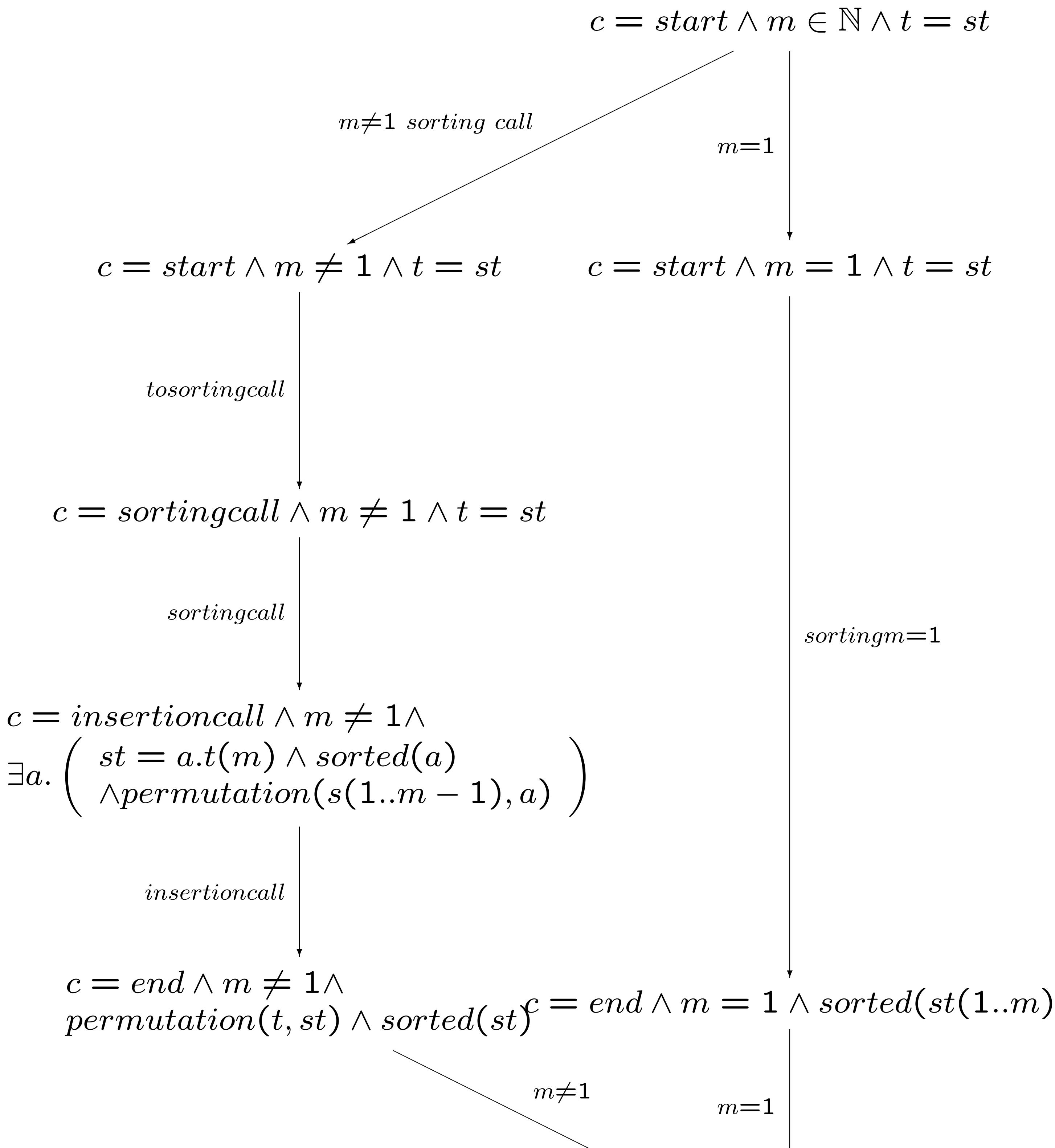
Problem 2: Sorting by insertion



- The problem is to sort an array t between 1 and m, where $dom(t) = 1..n$ and $m \leq n$.

- The sorting can be done by sorting the array from 1 to $m-1$ and then to insert the value $t(m)$ at the right position in $1..m$.

- The insertion event is considered as a call of procedure: the subproblem is solved in the same way by applying the guideline.


$c = start \wedge m \in \mathbb{N} \wedge t = st$
$m{\neq}1$ $sorting$ $call$
$m{=}1$
$c = start \wedge m \neq 1 \wedge t = st$
$c = start \wedge m = 1 \wedge t = st$
$tosortingcall$
$c = sortingcall \wedge m \neq 1 \wedge t = st$
$sortingcall$
$sortingm{=}1$
$c = insertioncall \wedge m \neq 1 \wedge$
$\exists a. \left(\begin{array}{l} st = a.t(m) \wedge sorted(a) \\ \wedge permutation(s(1..m-1), a) \end{array} \right)$
$insertioncall$
$c = end \wedge m \neq 1 \wedge$
$permutation(t, st) \wedge sorted(st)$
$c = end \wedge m = 1 \wedge sorted(st(1..m))$
$m{\neq}1$
$m{=}1$

Comments on Problem 2

- ◇ **The diagram gives the different events of the refinement model $IS2$; it contains an event called sortingcall, which is sorting the array t between the value 1 to $m-1$**

- ◇ **the event insertioncall which is inserting the value $t(m)$ at the right position in the array sorted between 1 and $m-1$.**

- ◇ **This last event can not be translated into an algorithmic expression and should be considered as defining a new problem which is the insertion of a value in a sorted array.**

- ◇ **We re-apply the guideline by starting a new development for solving the insertion problem.**

- ◇ **We use a diagram for illustrating the insertion of $t(m)$ into the values of $st(1..m-1)$.**

Problem 3: Floyd's algorithm

$$\begin{tikzcd}
\texttt{spath(l,a,b,g,D,FD)} \arrow[r, "call-as-event"] \arrow[d, "call"'] & \text{SPATH1} \arrow[r, "\text{SEES}"] \arrow[d, "\texttt{REFINEMENT}"'] & \text{CSPATH1} \arrow[d, "\texttt{EXTENDS}"'] \\
\texttt{spath} & \text{SPATH2} \arrow[l, "mapping"'] \arrow[r, "\text{SEES}"] & \text{CPATH2}
\end{tikzcd}$$

- `spath` is built from events of SPATH2
- `FD` states if the path exists
- `D` contains the cost of the minimal path, if it exists

```
/*   N = 1..n-1 */
void shortestpath (int l, int a, int b,int g[][n], int *D, int *FD)
{
  int D1,D2,D3,FD1,FD2,FD3;

  *FD = 0; FD1=0;FD2=0;FD3=0;
  if (l==0)
    {
      if (g[a][b] != NONE)
       { *FD = 1; *D = g[a][b];}
    }
  else
    {
       shortestpath(l-1,a,b,g,&D1,&FD1);
       if (FD1 == 1) {
          shortestpath(l-1,a,l,g,&D2,&FD2);
          if (FD2==1) {
            shortestpath(l-1,l,b,g,&D3,&FD3);
            if (FD3==1) {
            if (D1 < D2+D3)
              {*D= D1;}
            else
              {*D=D2+D3;};
            *FD = 1;}
            else
              {*D=D1;*FD=1;}}
          else
            {*D=D1;*FD=1;}}
       else
         {
           if  ( FD2 == 1 && FD3==1) {*D=D2+D3; *FD=1;}
       else
         {*FD=0;};}
    }}
```

Proof obligations

Model	Total	Auto	Manual	Reviewed	Undischarged
CSPATH1	8	8	0	0	0
SPATH1	5	4	1	0	0
SPATH2	493	317	176	0	0
Global	506	329	177	0	0

◇ **Proof Obligations are related to d.**

◇ **The prover provides an effective help for completing the invariant.**

Technical Justifications: defining traces

- Let M be an EVENT B machine and C a context seen by M.
- Let y be the list of variables of M,
- Let E be the set of events of M,
- let $Init(y)$ be the predicate defining the initial values of y in M.

The temporal framework of M is defined by the TLA specification denoted $\mathcal{S}pec(M)$:

$Init(y) \wedge \Box[Next]_y \wedge \mathrm{WF}_y(Next)$, where $Next \equiv \exists e \in E.BA(e)(y, y')$.

Technical Justifications: liveness properties

Suppose that PB is a context and PREPOST is a machine corresponding to a problem stating calls of a procedure. Suppose that the following diagram is validated:

$$\texttt{CALL} \xrightarrow{call-as-event} \text{PREPOST} \xrightarrow{\text{SEES}} \text{PB}$$

We assume that the preconditions are defined by P, i.e., $P_1, \ldots, P_n$, and the post-condition is defined by Q.

Then for any i in $1..n$, PREPOST satisfies $P_i(x,y) \leadsto Q(x,y)$.

Technical Justifications: refinement diagrams

A refinement diagram for M, P, and Q over L and A is an acyclic labelled graph over A with labels from G or E satisfying the following rules.

- There is a unique input node P with at least one outgoing arrow.
- There is a unique output node Q with no outgoing arrows.
- If R is related to S by a unique arrow labelled $e \in E$, then
  - It satisfies the property $R \leadsto S$
  - $\forall c, x, c', x'.R(c, x) \wedge I(M)(c, x) \wedge BA(e)(c, x, c', x') \Rightarrow S(c', x')$
  - $\forall c, x.R(c, x) \wedge I(M)(c, x) \Rightarrow \exists c', x'.BA(e)(c, x, c', x')$
  - If $R \equiv c = l1 \wedge A(x)$ and $S \equiv c = l2 \wedge B(x)$, then $l1 \neq l2$ and ℓ(R)=$l1$, ℓ(S)=$l2$.
- If R is related to $S_1, \ldots, S_p$, then
  - Each arrow R to S_i is labelled by a guard $g_i \in G$.
  - For any i in $1..p$ the following conditions hold.
    $$\left(\begin{array}{l} R \wedge I(M) \wedge g_i(x) \Rightarrow S_i \\ \forall j.j \in 1..p \wedge j \neq i \wedge R \wedge I(M) \wedge g_i(x) \Rightarrow \neg g_j(x) \end{array} \right.$$
  - $R \wedge I(M) \Rightarrow \exists i \in 1..p.g_i$.
- For each $e \in E$, there is only one instance of e in the diagram.

We use $PRE(D)$ for P and $POST(D)$ for Q.

Technical Justifications: a simple refinement diagram

$$
\begin{tikzcd}
& c = start \wedge P \arrow[dl, "P_1"'] \arrow[dr, "P_n"] & \\
c = start \wedge P_1 \arrow[dr, "call_1"] & \cdots & c = start \wedge P_n \arrow[dl, "call_n"] \\
& c = end \wedge Q &
\end{tikzcd}
$$

Technical Justifications: refinement diagram and liveness

Let M be a machine and let $D = (A, C, M, P, Q, G, E)$ be a refinement diagram for M.

1. If M satisfies $P \leadsto Q$ and $Q \leadsto R$, it satisfies $P \leadsto R$.
2. If M satisfies $P \leadsto Q$ and $R \leadsto Q$, it satisfies $(P \vee R) \leadsto Q$.
3. If I is invariant for M and if M satisfies $P \wedge I \leadsto Q$, then M satisfies $P \leadsto Q$.
4. If I is invariant for M and if M satisfies $P \wedge I \Rightarrow Q$, then M satisfies $P \leadsto Q$.
5. Let M be a machine and let $D = (A, C, M, P, Q, G, E)$ be a refinement diagram for M. If $P \xrightarrow{e} Q$ is a link of D for the machine M, then M satisfies $P \leadsto Q$.
6. Let M be a machine, and let $D = (A, C, M, P, Q, G, E)$ be a refinement diagram for M. If P and Q are two nodes of D such that there is a path in D from P to Q and any path from P can be extended in a path containing Q, then M satisfies $P \leadsto Q$.

Let M be a machine and let $D = (A, C, M, P, Q, G, E)$ be a refinement diagram for M.
Then M satisfies $(c = start \wedge PRE(D)) \leadsto (c = end \wedge POST(D))$.

Concluding Remarks and Futur Works

- ◇ Automatic process for producing an algorithm from the EVENT B models.
- ◇ Management of problems and subproblems
- ◇ Systematic way to develop a sequential algorithm
- ◇ Extending to concurrent/distributed algorithms
- ◇ Plugin for the translation
- ◇ Current works: cryptologic algorithms, access control systems, distributed algorithms, . . .
- ◇ Case studies: next slides