Modelling Software-based Systems

Correctness by Contruction of sequential algorithms using Event-B Refinement by iterative patterns

Master Informatique

Dominique Méry
Telecom Nancy,Université de Lorraine

23 novembre 2026

dominique.mery@loria.fr

Summary

1. CM5
2. Overview
3. Programming by contract
   - Short view
   - The call-as-event idea
   - Main idea
4. Strategy 1 : Transformations of the ALGO machine
5. Strategy 2 : Internalizing events by merging
6. Analysis and then Synthesis
   - Analysis using Refinement
   - Synthesis by Merging
7. Summing a list of integer values
8. Stategy 3 : The Recursive Interpretation of call
9. Two examples
   - Computing the square of a number
   - Searching a key in an array

Current Section Summary

1. CM5
2. Overview
3. Programming by contract
   - Short view
   - The call-as-event idea
   - Main idea
4. Strategy 1 : Transformations of the ALGO machine
5. Strategy 2 : Internalizing events by merging
6. Analysis and then Synthesis
   - Analysis using Refinement
   - Synthesis by Merging
7. Summing a list of integer values
8. Stategy 3 : The Recursive Interpretation of call
9. Two examples
   - Computing the square of a number
   - Searching a key in an array

Current Section Summary

1. CM5
2. Overview
3. Programming by contract
   - Short view
   - The call-as-event idea
   - Main idea
4. Strategy 1 : Transformations of the ALGO machine
5. Strategy 2 : Internalizing events by merging
6. Analysis and then Synthesis
   - Analysis using Refinement
   - Synthesis by Merging
7. Summing a list of integer values
8. Stategy 3 : The Recursive Interpretation of call
9. Two examples
   - Computing the square of a number
   - Searching a key in an array

Introduction

Introduction

- Guidelines for using the transformations proposed b y J.-R. Abrial and used for generating sequential algorithms.

Introduction

- Guidelines for using the transformations proposed b y J.-R. Abrial and used for generating sequential algorithms.
- Possibility to combine with others provers as TLAPS, Frama-c

Introduction

- Guidelines for using the transformations proposed b y J.-R. Abrial and used for generating sequential algorithms.
- Possibility to combine with others provers as TLAPS, Frama-c
- Illustrating the merging of events and the step to code.

Introduction

- Guidelines for using the transformations proposed b y J.-R. Abrial and used for generating sequential algorithms.
- Possibility to combine with others provers as TLAPS, Frama-c
- Illustrating the merging of events and the step to code.
- Relationship to PlusCal and TLA$^+$ proof tools

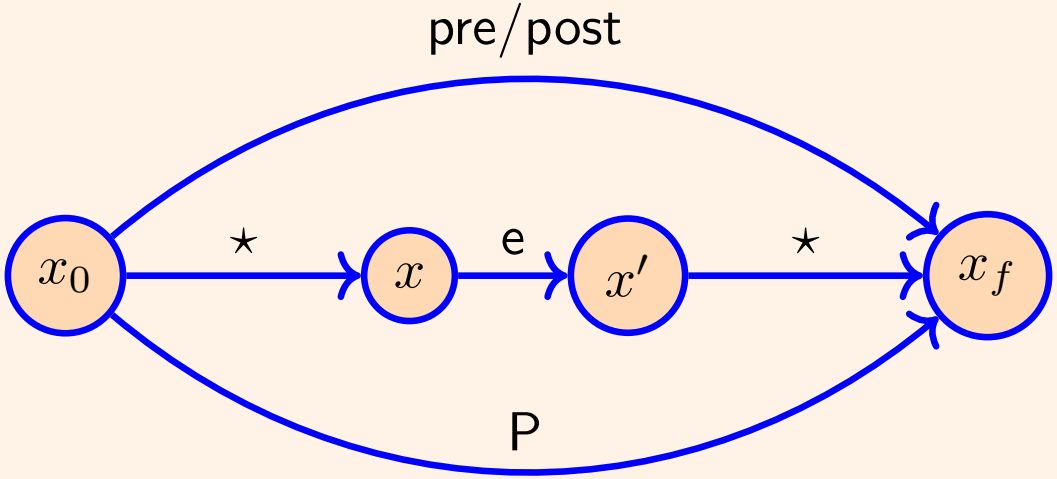
Introduction

- Guidelines for using the transformations proposed b y J.-R. Abrial and used for generating sequential algorithms.
- Possibility to combine with others provers as TLAPS, Frama-c
- Illustrating the merging of events and the step to code.
- Relationship to PlusCal and TLA$^+$ proof tools
- Possibility to derive algorithms and C programs.

Big steps versus Small steps

Frpm pre/post to algorithm(s)

Refining a pre/post specification into a sequential algorithm



Iterative patterns

- (1) Iterative Pattern I : Transformations from machine ALGO into an algorithm following rules of J.-R. Abrial.
- (2) Iterative Pattern II : Transformations from machine MERGE into an algorithm following rules of transformations of primed relation defining the unique event of the last machine MERGE
- (3) Recursive Pettern : One step Refinement Process from PREPOST and transformations of
- Target languages as C, PlusCal.

Current Section Summary

1. CM5
2. Overview
3. Programming by contract
   - Short view
   - The call-as-event idea
   - Main idea
4. Strategy 1 : Transformations of the ALGO machine
5. Strategy 2 : Internalizing events by merging
6. Analysis and then Synthesis
   - Analysis using Refinement
   - Synthesis by Merging
7. Summing a list of integer values
8. Stategy 3 : The Recursive Interpretation of call
9. Two examples
   - Computing the square of a number
   - Searching a key in an array

Current Sub Section Summary

1. CM5
2. Overview
3. Programming by contract
   - Short view
   - The call-as-event idea
   - Main idea
4. Strategy 1 : Transformations of the ALGO machine
5. Strategy 2 : Internalizing events by merging
6. Analysis and then Synthesis
7. Summing a list of integer values
8. Stategy 3 : The Recursive Interpretation of call
9. Two examples

Verification of contract (I)

A program P *satisfies* a contract (pre,post) :

- P transforms a variable x from an initial value x_0 and produces a final value x_f : $x_0 \xrightarrow{P} x_f$
- x_0 satisfait pre : pre(x_0) and x_f satisfait post : post(x_0, x_f)
- pre$(x_0) \wedge x_0 \xrightarrow{P} x_f \Rightarrow$ post(x_0, x_f)

requires $pre(x_0)$
ensures $post(x_0, x_f)$
variables X

begin
$0 : P_0(x_0, x)$
instruction$_0$
...
$i : P_i(x_0, x)$
...
instruction$_{f-1}$
$f : P_f(x_0, x)$
end

Floyd style

- $pre(x_0) \wedge x = x_0 \Rightarrow P_0(x_0, x)$
- $pre(x_0) \wedge P_f(x_0, x) \Rightarrow post(x_0, x)$
- For each pair ℓ, ℓ' such that $\ell \longrightarrow \ell'$, one checks that foa any value $x, x' \in \text{MEMORY}$

$$\left(\begin{array}{l} \left(\begin{array}{l} pre(x_0) \wedge P_\ell(x_0, x)) \\ \wedge cond_{\ell,\ell'}(x) \wedge x' = f_{\ell,\ell'}(x) \end{array} \right) \\ \Rightarrow P_{\ell'}(x_0, x') \end{array} \right),$$

Verification du contract (II)

A program P *satisfies* a contract (pre,post) :

- P transforms a variable x from an initial value x_0 and produces a final value x_f : $x_0 \xrightarrow{P} x_f$
- x_0 satisfait pre : pre(x_0) and x_f satisfait post : post(x_0, x_f)
- pre$(x_0) \wedge x_0 \xrightarrow{P} x_f \Rightarrow$ post(x_0, x_f)

Hoare style

requires $pre(x_0)$
ensures $post(x_0, x_f)$
variables X

begin
$0 : P_0(x_0, x)$
$\texttt{instruction}_0$
...
$i : P_i(x_0, x)$
...
$\texttt{instruction}_{f-1}$
$f : P_f(x_0, x)$
end

Verification du contract (II)

A program P *satisfies* a contract (pre,post) :

- P transforms a variable x from an initial value x_0 and produces a final value x_f : $x_0 \xrightarrow{P} x_f$
- x_0 satisfait pre : pre(x_0) and x_f satisfait post : post(x_0, x_f)
- pre$(x_0) \wedge x_0 \xrightarrow{P} x_f \Rightarrow$ post(x_0, x_f)

requires $pre(x_0)$
ensures $post(x_0, x_f)$
variables X
begin
$0 : P_0(x_0, x)$
$\texttt{instruction}_0$
...
$i : P_i(x_0, x)$
...
$\texttt{instruction}_{f-1}$
$f : P_f(x_0, x)$
end

Hoare style

- $\forall x_f, x_0.$pre$(x_0) \wedge x_0 \xrightarrow{P} x_f \Rightarrow$ post(x_0, x_f)

Verification du contract (II)

A program P *satisfies* a contract (pre,post) :

- P transforms a variable x from an initial value x_0 and produces a final value x_f : $x_0 \xrightarrow{P} x_f$
- x_0 satisfait pre : pre(x_0) and x_f satisfait post : post(x_0, x_f)
- pre$(x_0) \wedge x_0 \xrightarrow{P} x_f \Rightarrow$ post(x_0, x_f)

requires $pre(x_0)$
ensures $post(x_0, x_f)$
variables X
begin
$0 : P_0(x_0, x)$
$\texttt{instruction}_0$
...
$i : P_i(x_0, x)$
...
$\texttt{instruction}_{f-1}$
$f : P_f(x_0, x)$
end

Hoare style

- $\forall x_f, x_0.$pre$(x_0) \wedge x_0 \xrightarrow{P} x_f \Rightarrow$ post(x_0, x_f)
- $\forall x_f, x_0.$pre$(x_0) \Rightarrow (x_0 \xrightarrow{P} x_f \Rightarrow$ post$(x_0, x_f))$

Verification du contract (II)

A program P *satisfies* a contract (pre,post) :

- P transforms a variable x from an initial value x_0 and produces a final value x_f : $x_0 \xrightarrow{P} x_f$
- x_0 satisfait pre : pre(x_0) and x_f satisfait post : post(x_0, x_f)
- pre$(x_0) \wedge x_0 \xrightarrow{P} x_f \Rightarrow$ post(x_0, x_f)

requires $pre(x_0)$
ensures $post(x_0, x_f)$
variables X
begin
$0 : P_0(x_0, x)$
$\texttt{instruction}_0$
...
$i : P_i(x_0, x)$
...
$\texttt{instruction}_{f-1}$
$f : P_f(x_0, x)$
end

Hoare style

- $\forall x_f, x_0.$pre$(x_0) \wedge x_0 \xrightarrow{P} x_f \Rightarrow$ post(x_0, x_f)
- $\forall x_f, x_0.$pre$(x_0) \Rightarrow (x_0 \xrightarrow{P} x_f \Rightarrow$ post$(x_0, x_f))$
- $\forall x_0.$pre$(x_0) \Rightarrow \forall x_f.(x_0 \xrightarrow{P} x_f \Rightarrow$ post$(x_0, x_f))$

Verification du contract (II)

A program P *satisfies* a contract (pre,post) :

- P transforms a variable x from an initial value x_0 and produces a final value x_f : $x_0 \xrightarrow{P} x_f$
- x_0 satisfait pre : pre(x_0) and x_f satisfait post : post(x_0, x_f)
- pre$(x_0) \wedge x_0 \xrightarrow{P} x_f \Rightarrow$ post(x_0, x_f)

requires $pre(x_0)$
ensures $post(x_0, x_f)$
variables X
begin
$0 : P_0(x_0, x)$
$\texttt{instruction}_0$
...
$i : P_i(x_0, x)$
...
$\texttt{instruction}_{f-1}$
$f : P_f(x_0, x)$
end

Hoare style

- $\forall x_f, x_0.$pre$(x_0) \wedge x_0 \xrightarrow{P} x_f \Rightarrow$ post(x_0, x_f)
- $\forall x_f, x_0.$pre$(x_0) \Rightarrow (x_0 \xrightarrow{P} x_f \Rightarrow$ post$(x_0, x_f))$
- $\forall x_0.$pre$(x_0) \Rightarrow \forall x_f.(x_0 \xrightarrow{P} x_f \Rightarrow$ post$(x_0, x_f))$
- $\forall x_0.$pre$(x_0) \Rightarrow \forall x.(x_0 \xrightarrow{P} x \Rightarrow$ post$(x_0, x))$

Verification du contract (II)

A program P *satisfies* a contract (pre,post) :

- P transforms a variable x from an initial value x_0 and produces a final value x_f : $x_0 \xrightarrow{P} x_f$
- x_0 satisfait pre : pre(x_0) and x_f satisfait post : post(x_0, x_f)
- pre$(x_0) \wedge x_0 \xrightarrow{P} x_f \Rightarrow$ post(x_0, x_f)

requires $pre(x_0)$
ensures $post(x_0, x_f)$
variables X
begin
$0 : P_0(x_0, x)$
$\texttt{instruction}_0$
...
$i : P_i(x_0, x)$
...
$\texttt{instruction}_{f-1}$
$f : P_f(x_0, x)$
end

Hoare style

- $\forall x_f, x_0.$pre$(x_0) \wedge x_0 \xrightarrow{P} x_f \Rightarrow$ post(x_0, x_f)
- $\forall x_f, x_0.$pre$(x_0) \Rightarrow (x_0 \xrightarrow{P} x_f \Rightarrow$ post$(x_0, x_f))$
- $\forall x_0.$pre$(x_0) \Rightarrow \forall x_f.(x_0 \xrightarrow{P} x_f \Rightarrow$ post$(x_0, x_f))$
- $\forall x_0.$pre$(x_0) \Rightarrow \forall x.(x_0 \xrightarrow{P} x \Rightarrow$ post$(x_0, x))$
- $\forall x_0.$pre$(x_0) \Rightarrow WLP(P)($post$(x_0, x))$

A Simple C Function

Listing 1 – Simple contract

```
#ifndef _A_H
#define _A_H
// Definition of the  mathematical function mathpower2
/*@ axiomatic auxmath {
  @ lemma rule1: \forall int n; n >0 ==> n*n == (n-1)*(n-1)+2*(n-1)+1;
  @ } */
/*@  requires  0 <= x;
  @ requires x*x <= INT_MAX;
     assigns \nothing;
     ensures \result == x*x;
*/
int power2(int x);
#endif
```

A Simple C Function

Listing 2 – Simple contract

```
#include <limits.h>
#include "power2.h"

int power2(int x)
{int r,k,cv,cw,or,ok,ocv,ocw;
  r=0;k=0;cv=0;cw=0;or=0;ok=k;ocv=cv;ocw=cw;
     /*@ loop invariant 0 <= cv && 0 <= cw && 0 <= k;
       @ loop invariant cv == k*k;
        @ loop invariant k <= x;
        @ loop invariant cw == 2*k;
        @ loop invariant 4*cv == cw*cw;
        @ loop assigns k,cv,cw,or,ok,ocv,ocw;
       @ loop variant x-k;
*/
  while (k<x)
    {
    ok=k;ocv=cv;ocw=cw;
      k=ok+1;
      cv=ocv+ocw+1;
      cw=ocw+2;

    }
  r=cv;
  return(r);
}
```

A Simple C Function

Listing 3 – Simple contract

```
frama-c -wp -rte power2.c
[kernel] Parsing power2.c (with preprocessing)
[rte:annot] annotating function power2
[wp] 30 goals scheduled
[wp] Proved goals:    32 / 32
  Terminating:         1
  Unreachable:         1
  Qed:                22 (5ms-1ms-7ms)
  Alt-Ergo 2.6.0:      8 (13ms-15ms-18ms)
```

Correct by Construction

- Finding or computing annotations is difficult and even undecidable !

Correct by Construction

- Finding or computing annotations is difficult and even undecidable !
- Given

Contract EX
Variables $X(int)$
Requires $pre(x_0)$
Ensures $post(x_0, x_f)$

Correct by Construction

- Finding or computing annotations is difficult and even undecidable !
- Given

  Contract EX
  Variables $X(int)$
  Requires $pre(x_0)$
  Ensures $post(x_0, x_f)$

- Design of the algorithm ALG fulfiling the contract

Correct by Construction

- Finding or computing annotations is difficult and even undecidable !
- Given

Contract EX
Variables $X(int)$
Requires $pre(x_0)$
Ensures $post(x_0, x_f)$

- Design of the algorithm ALG fulfiling the contract
- HOARE triple :

$$\{\mathbf{pre(x_0)} \wedge \mathbf{x} = \mathbf{x_0}\}\text{ALG}\{\mathbf{post(x_0, x)}\}$$

Correct by Construction

- Finding or computing annotations is difficult and even undecidable !
- Given

Contract EX
Variables $X(int)$
Requires $pre(x_0)$
Ensures $post(x_0, x_f)$

- Design of the algorithm ALG fulfiling the contract
- HOARE triple :

$$\{\mathbf{pre(x_0)} \wedge \mathbf{x} = \mathbf{x_0}\}\text{ALG}\{\mathbf{post(x_0, x)}\}$$

Idea ?

From Contract EX, using a step by step approach to find an algorithm ALG satisfying it using refinement and Event-B models.
Applying the CLEANROOM model but with tool support

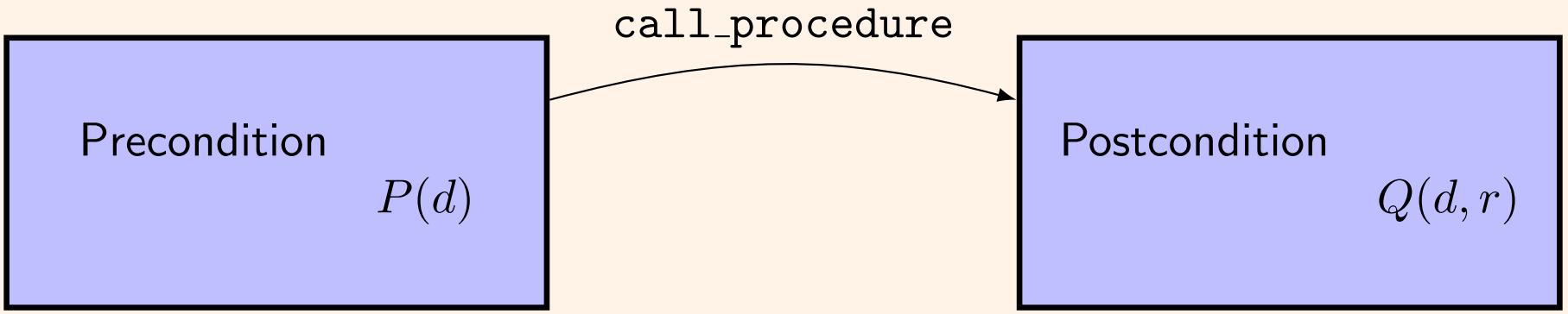
Current Sub Section Summary

1. CM5
2. Overview
3. Programming by contract
   - Short view
   - The call-as-event idea
   - Main idea
4. Strategy 1 : Transformations of the ALGO machine
5. Strategy 2 : Internalizing events by merging
6. Analysis and then Synthesis
7. Summing a list of integer values
8. Stategy 3 : The Recursive Interpretation of call
9. Two examples

Topics

- Providing patterns for helping the use of refinement.
- Making proofs as simple as possible.
- Simplifying the discovery of invariants
- we focus on the transformation of the final concrete specification into a PlusCal algorithm :
  1. transforming an EVENT B specification into a PlusCal algorithm
  2. using the merging of events to get only one event corresponding to the call of the algroithm
  3. double checking the resulting algorithm by using TLA$^+$ and its proof system.

The call as event idea



- `call_procedure` is simulated by the event call_procedure :

EVENT CALL_PROCEDURE
WHEN
$P(d)$
THEN
$r :| Q(d, r')$
END

- $P(d)$ is specifying the inputs of the procedure.
- $Q(d, r)$ is expressing the relation between the inputs d and the results r.

Current Sub Section Summary

1. CM5
2. Overview
3. Programming by contract
   - Short view
   - The call-as-event idea
   - Main idea
4. Strategy 1 : Transformations of the ALGO machine
5. Strategy 2 : Internalizing events by merging
6. Analysis and then Synthesis
7. Summing a list of integer values
8. Stategy 3 : The Recursive Interpretation of call
9. Two examples

Call as event

- A pre/post specification is a pair of assertions $(pre, post)$
- $(pre, post)$ is defining a *transformation* over variables that is correct with respect to *HOARE* triples :

$$\{pre\}transformation\{post\}$$

- **Problem** : Design of a program PROG simulating the *transformation* with respect to $(pre, post)$:

$$\{pre\}\text{PROG}\{post\}$$

- **Idea** : Writing a *one-shot* relation simulating the relation over *flexible* variables :

$$\forall x, y.pre(x) \wedge R(x, y) \Rightarrow post(x, y)$$

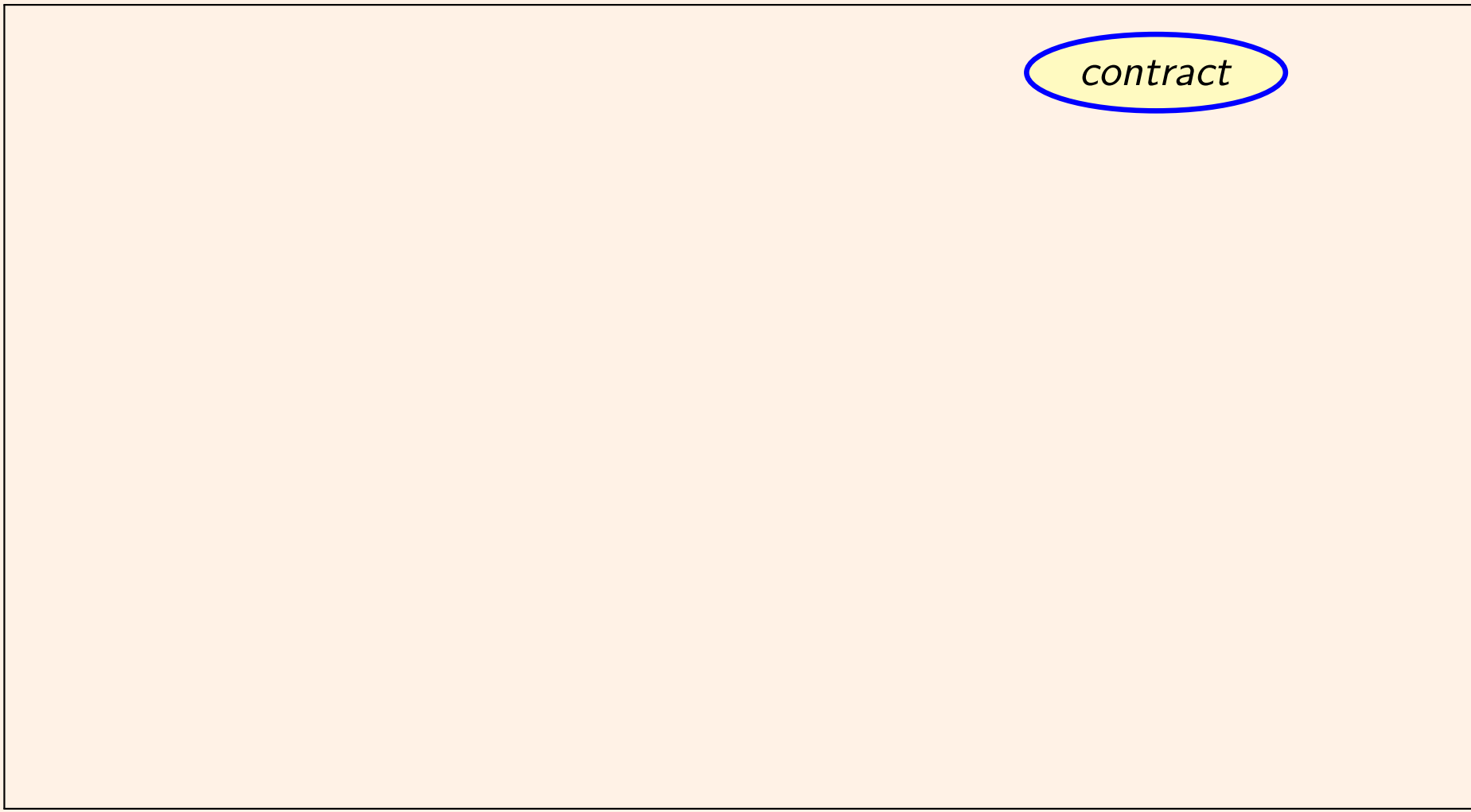
- **event** : Expressing the relation :

EVENT PROG
WHEN
$pre(x)$
THEN
$y :| post(x, y')$
END

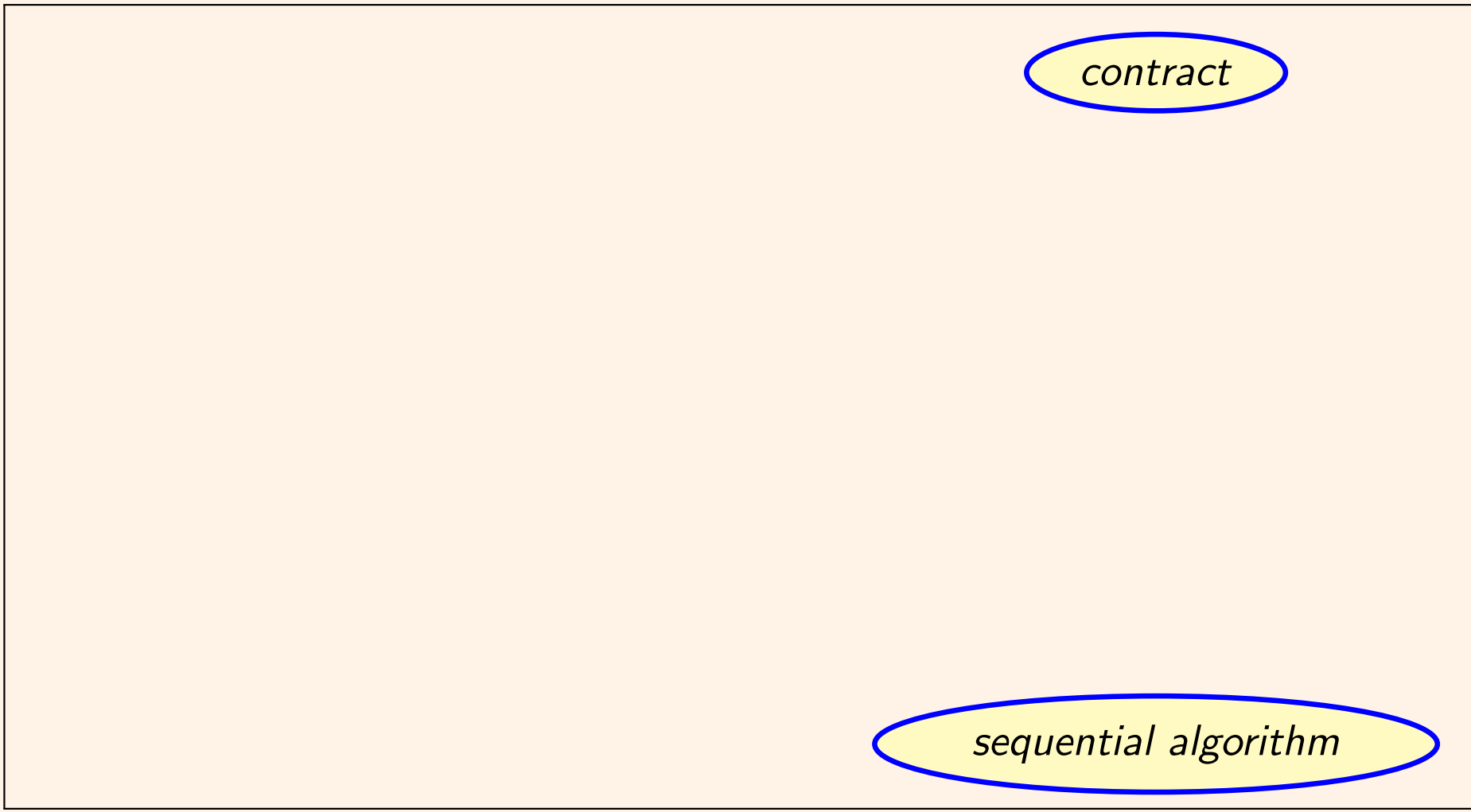
Current Section Summary

1. CM5
2. Overview
3. Programming by contract
   - Short view
   - The call-as-event idea
   - Main idea
4. Strategy 1 : Transformations of the ALGO machine
5. Strategy 2 : Internalizing events by merging
6. Analysis and then Synthesis
   - Analysis using Refinement
   - Synthesis by Merging
7. Summing a list of integer values
8. Stategy 3 : The Recursive Interpretation of call
9. Two examples
   - Computing the square of a number
   - Searching a key in an array

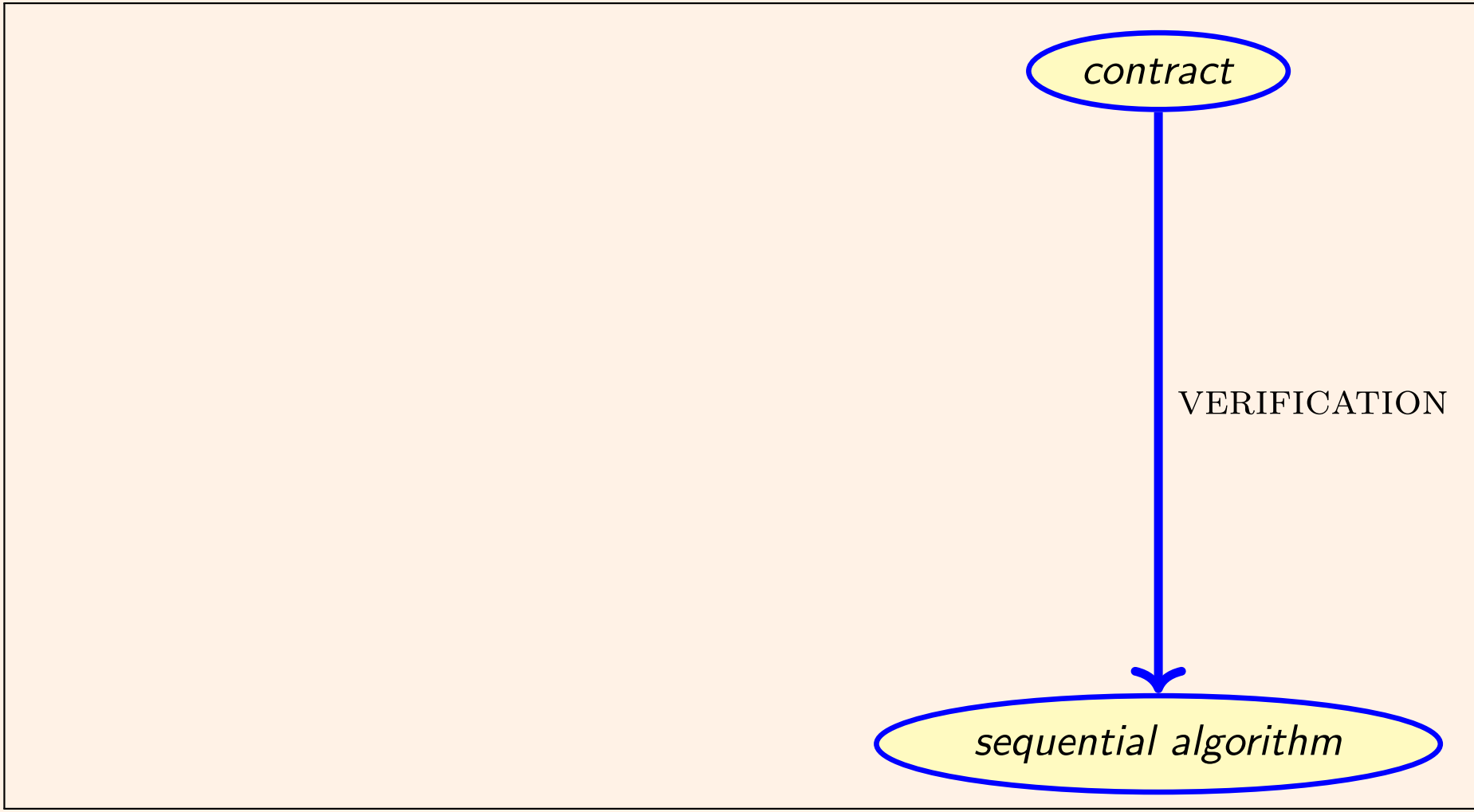
The sequential pattern



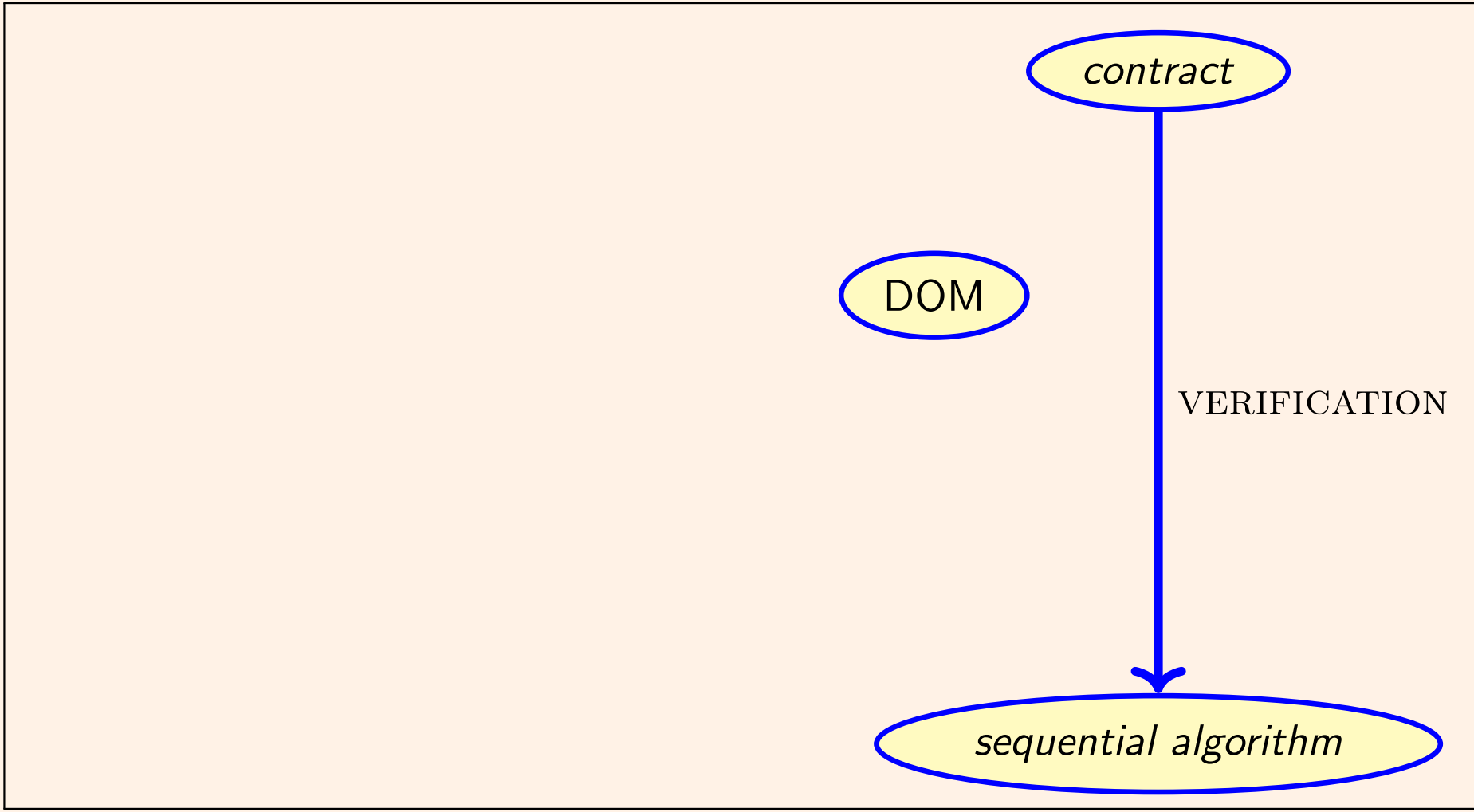
The sequential pattern



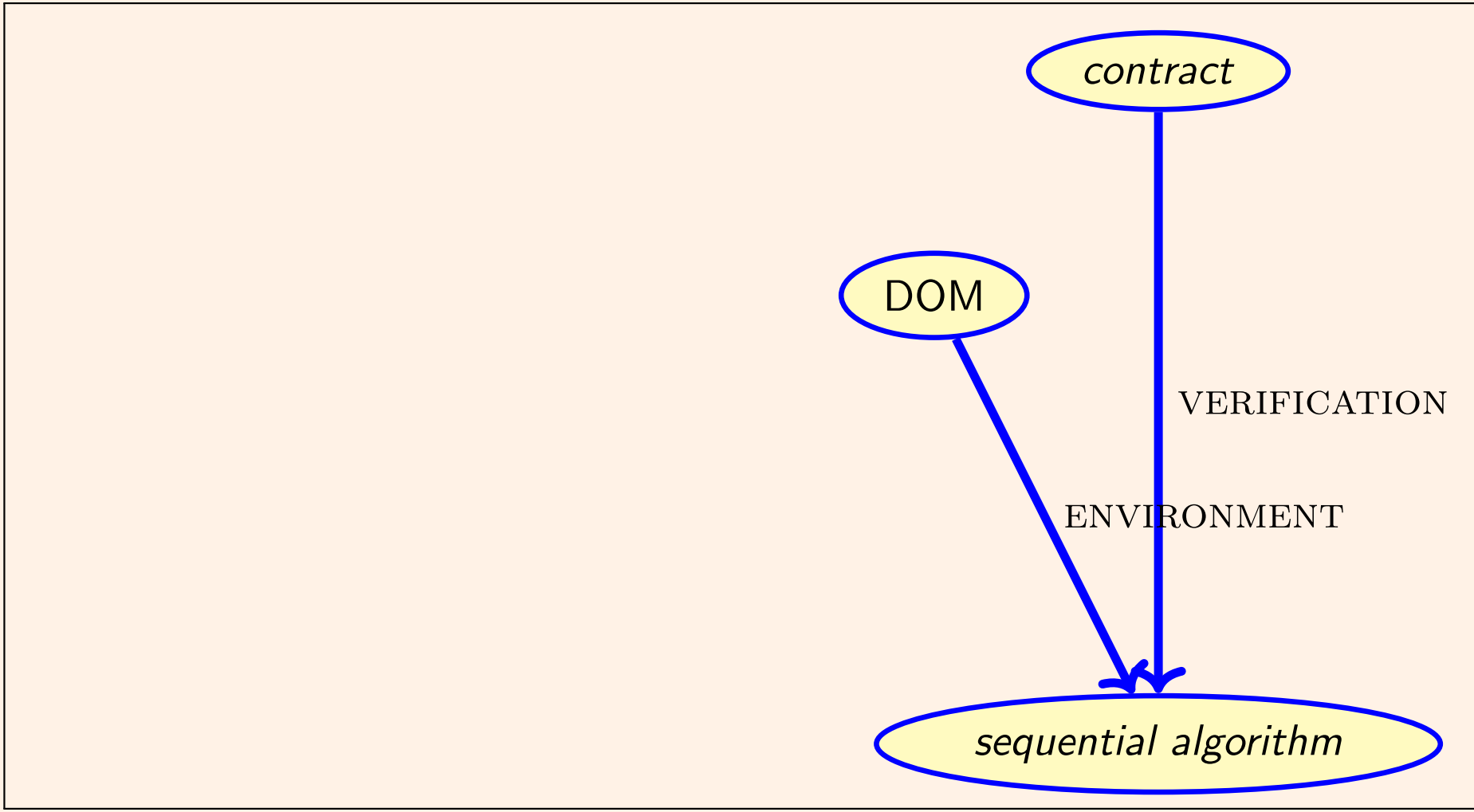
The sequential pattern



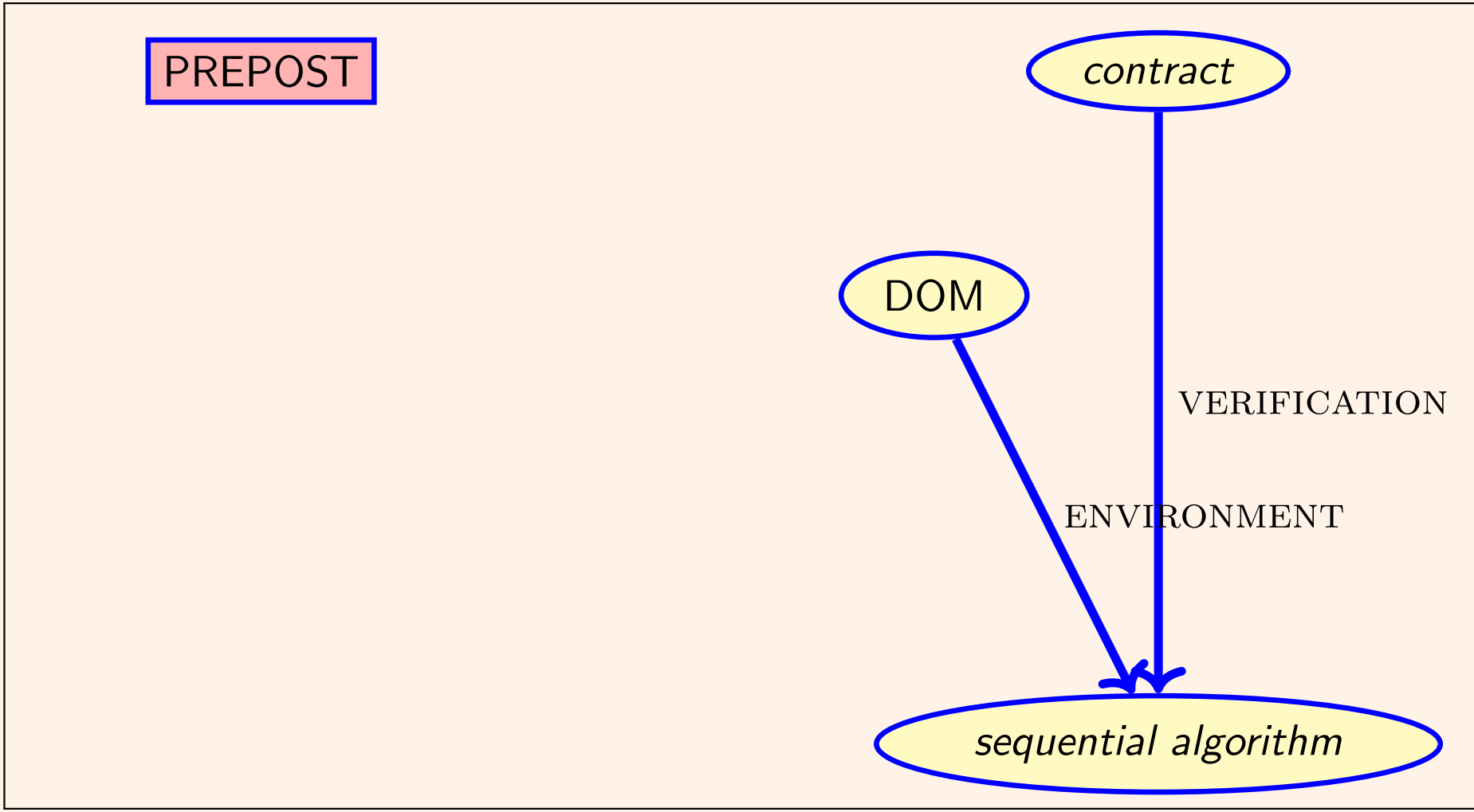
The sequential pattern



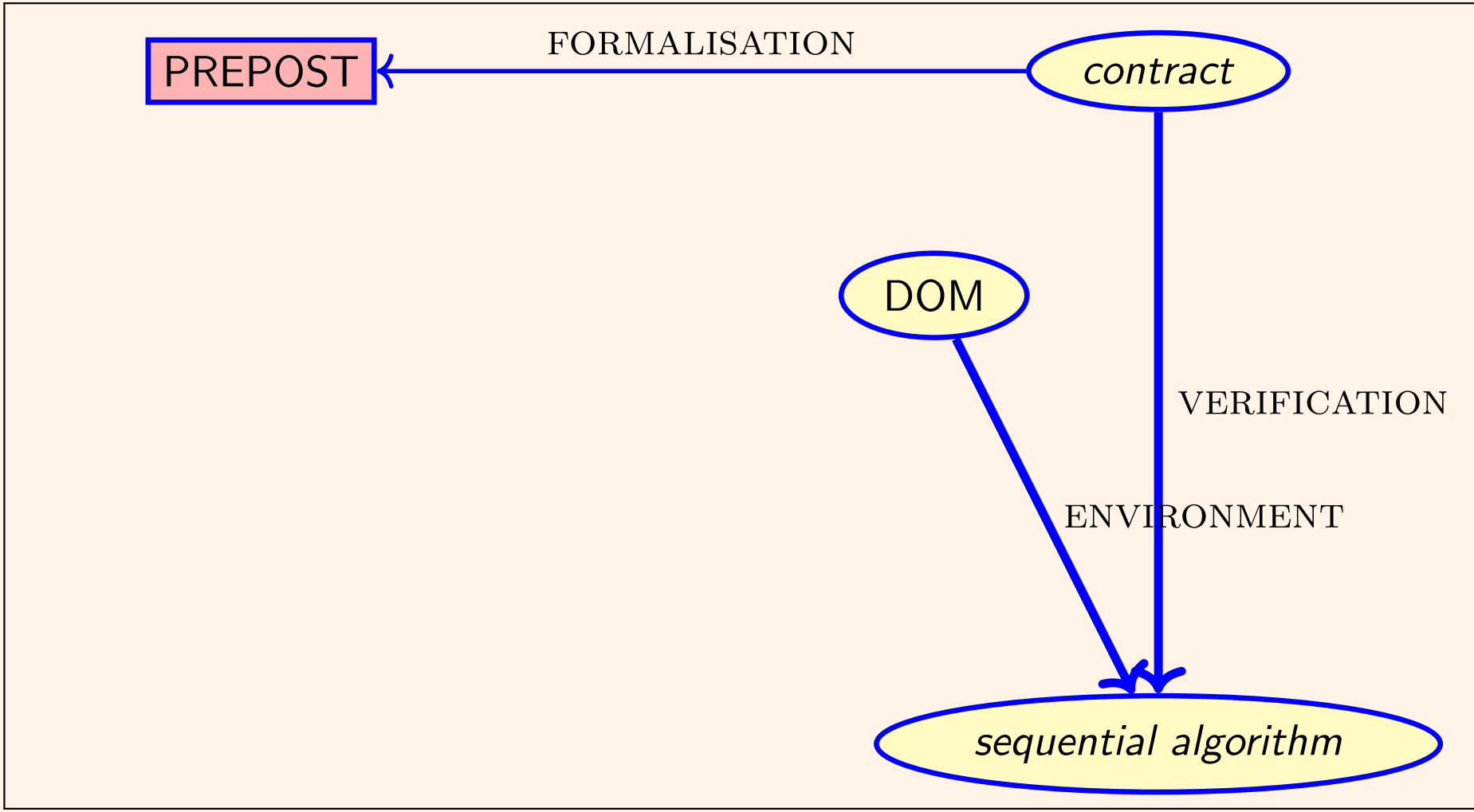
The sequential pattern



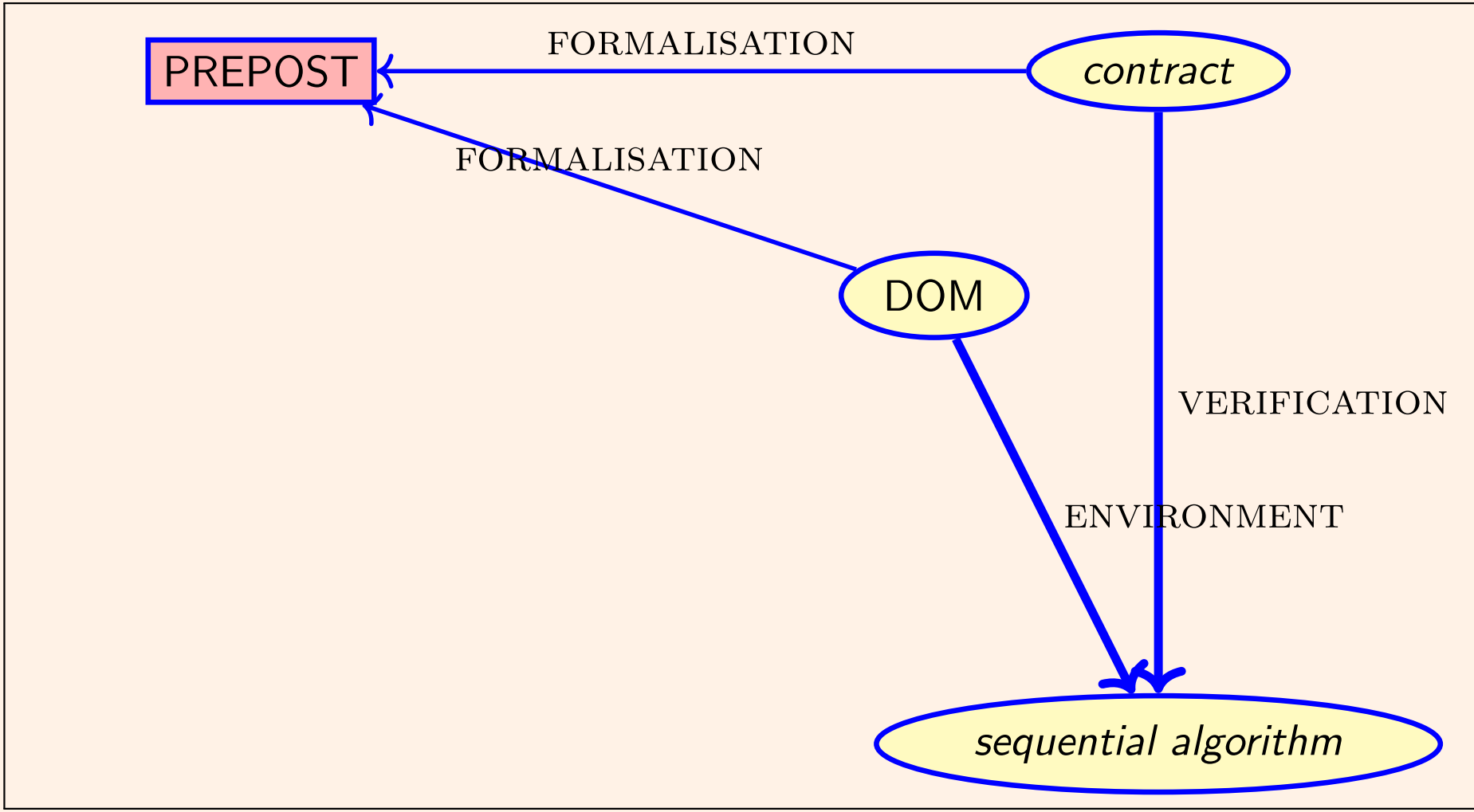
The sequential pattern



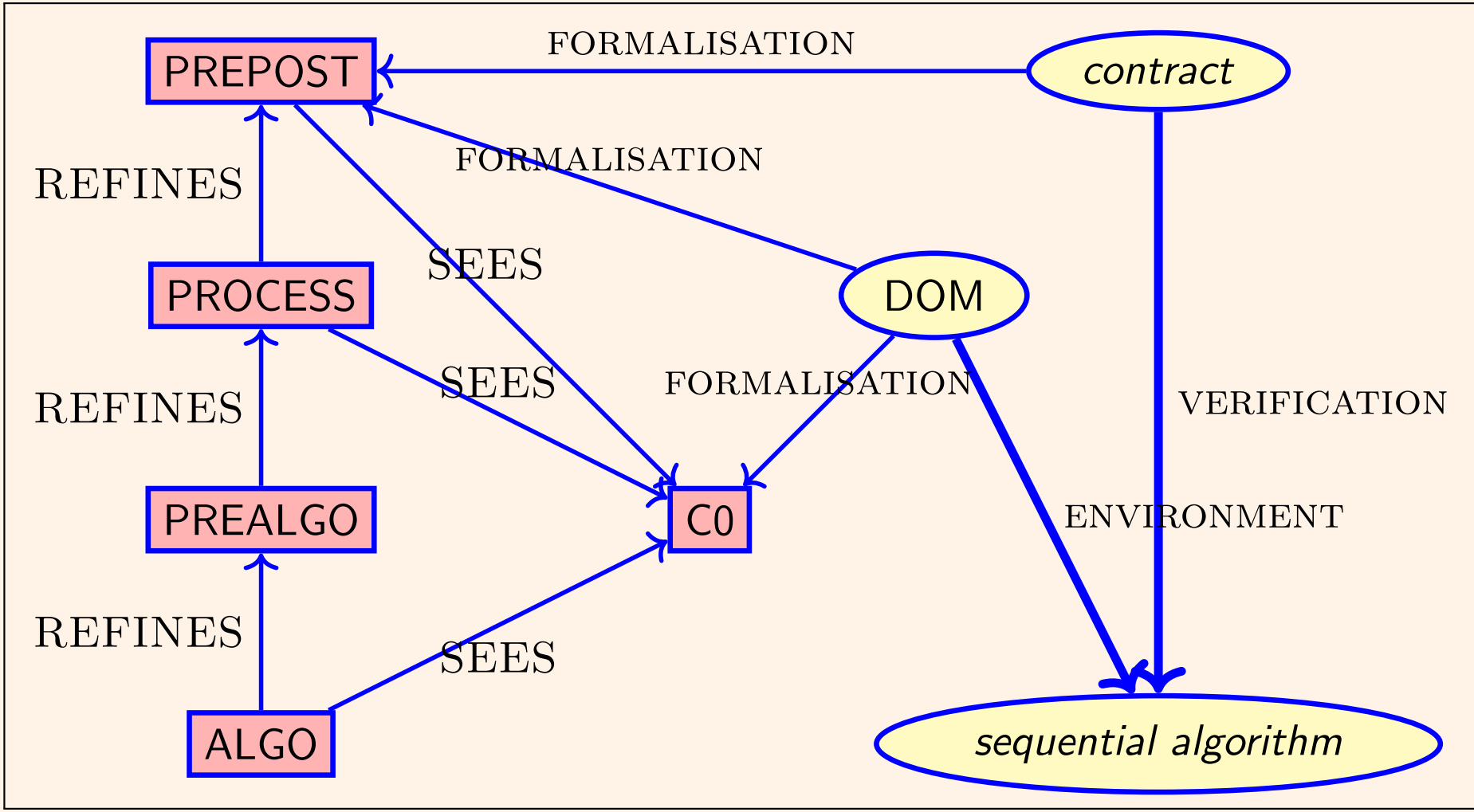
The sequential pattern



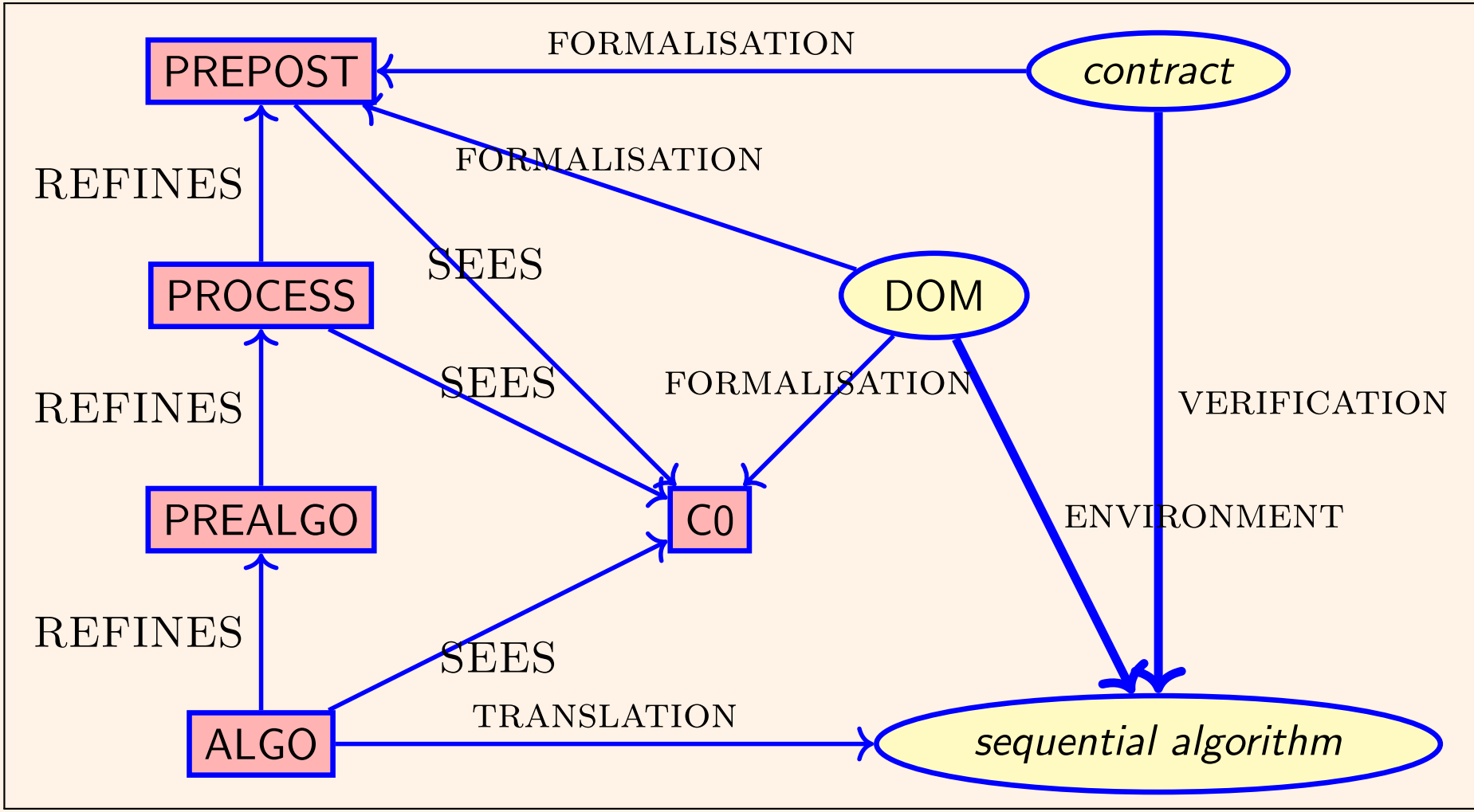
The sequential pattern



The sequential pattern



The sequential pattern



First steps of refinement

- Translating the contract into a machine called PREPOST and expressing either one event observing the call of the hidden function, or events corresponding to the possible conditions of call.
- Defining mathematical properties of required data in a shared context.

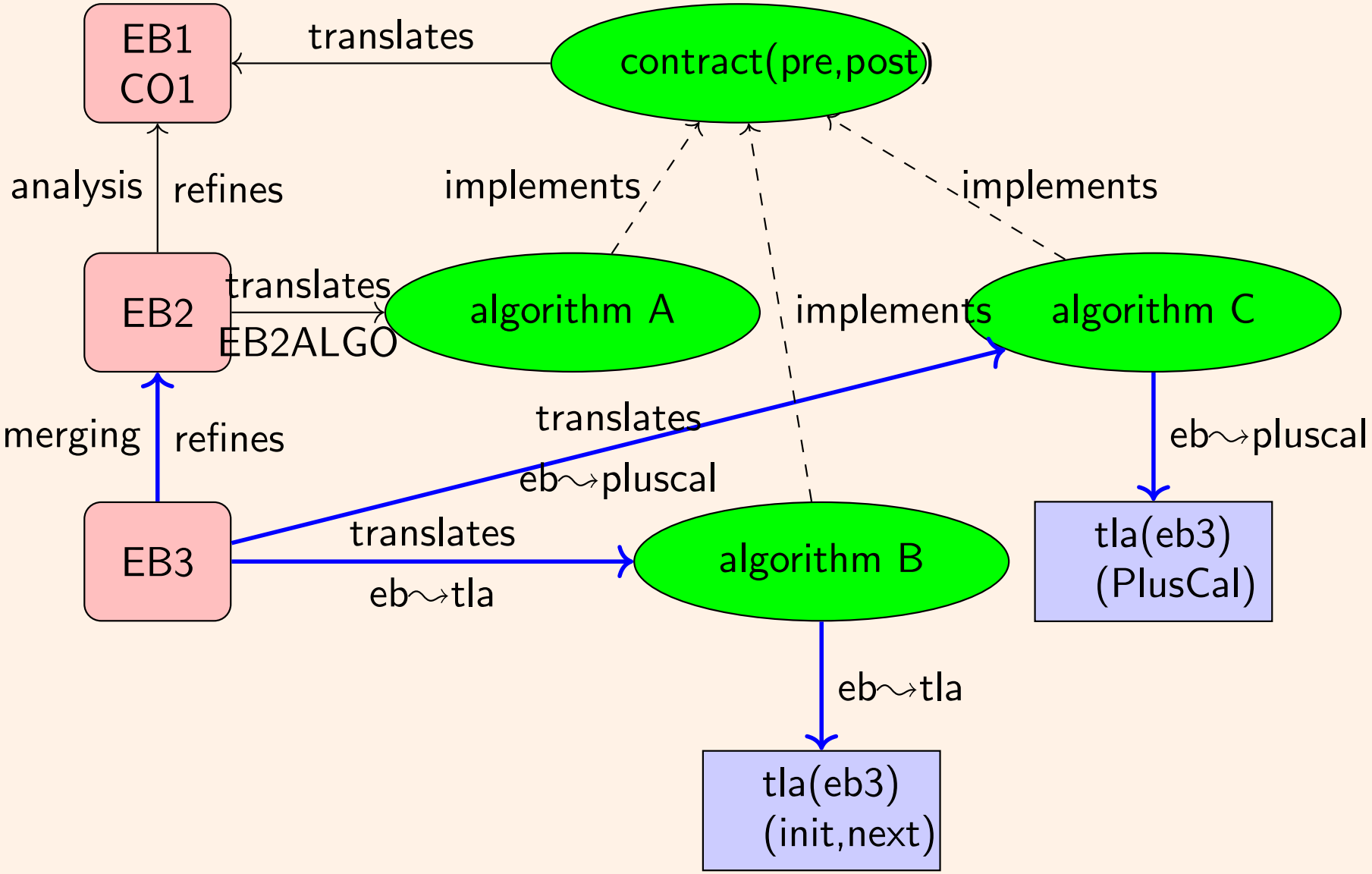
Problem squares

Values	$\mathbb{N}$
First term	$v_0 = 0$
Recurrence	$\forall n \in \mathbb{N} : v_{n+1} = v_n + w_n + 1$
First term	$w_0 = 0$
Recurrence	$\forall n \in \mathbb{N} : w_{n+1} = w_n + 2$
First term	$u_0 = 0$
Recurrence	$\forall n \in \mathbb{N} : u_{n+1} = u_n + 1$
Property	$\forall i \in 0 \mathinner{..} n_0 \cdot u_i = i$
Property	$\forall i \in 0 \mathinner{..} n_0 \cdot w_i = 2i$

Next steps of refinement

- From the sequence u used for defining the result, refining the machine PREPOST into a machine COMPUTING filling a variable denoted uu and storing the sequence u into the variable uu by introducing new events.
- Refining the machine COMPUTING into the machine PREALGO by simplifying the conditioons and making them computable.
- Progressing by refinement of the machine PREALGO into ALGO and hiding the model variables and keeping the computer variables.
- The machine ALGO is the machine on which the rules defined by Abrial are applied and applicable.

Global Picture



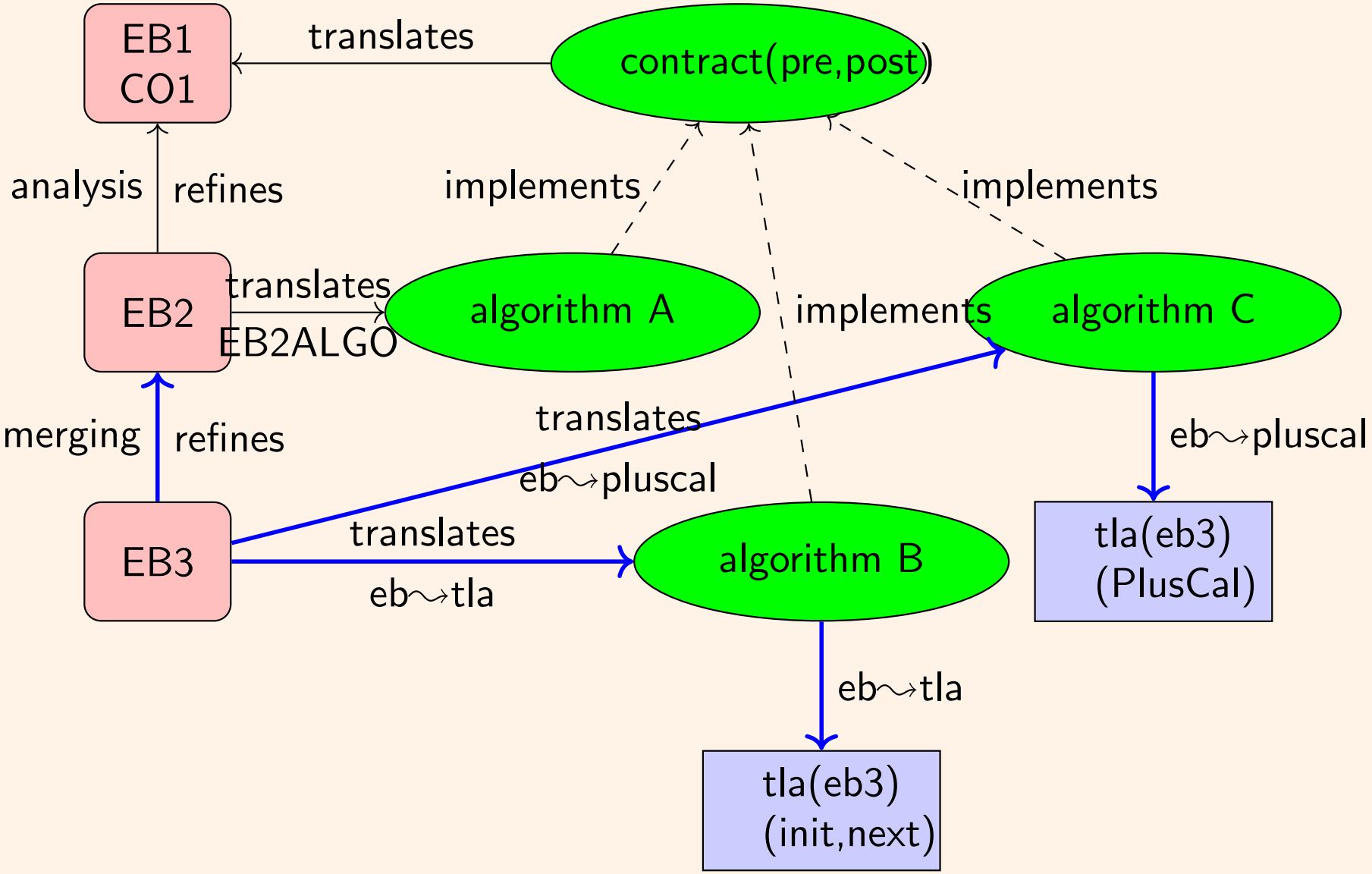
Strategy I

- Producing programs or algorithms from the last refinement machine ALFO.
- Two main transformation rules are used.
- The EB2ALGO plugin is the handling the strategy 1.

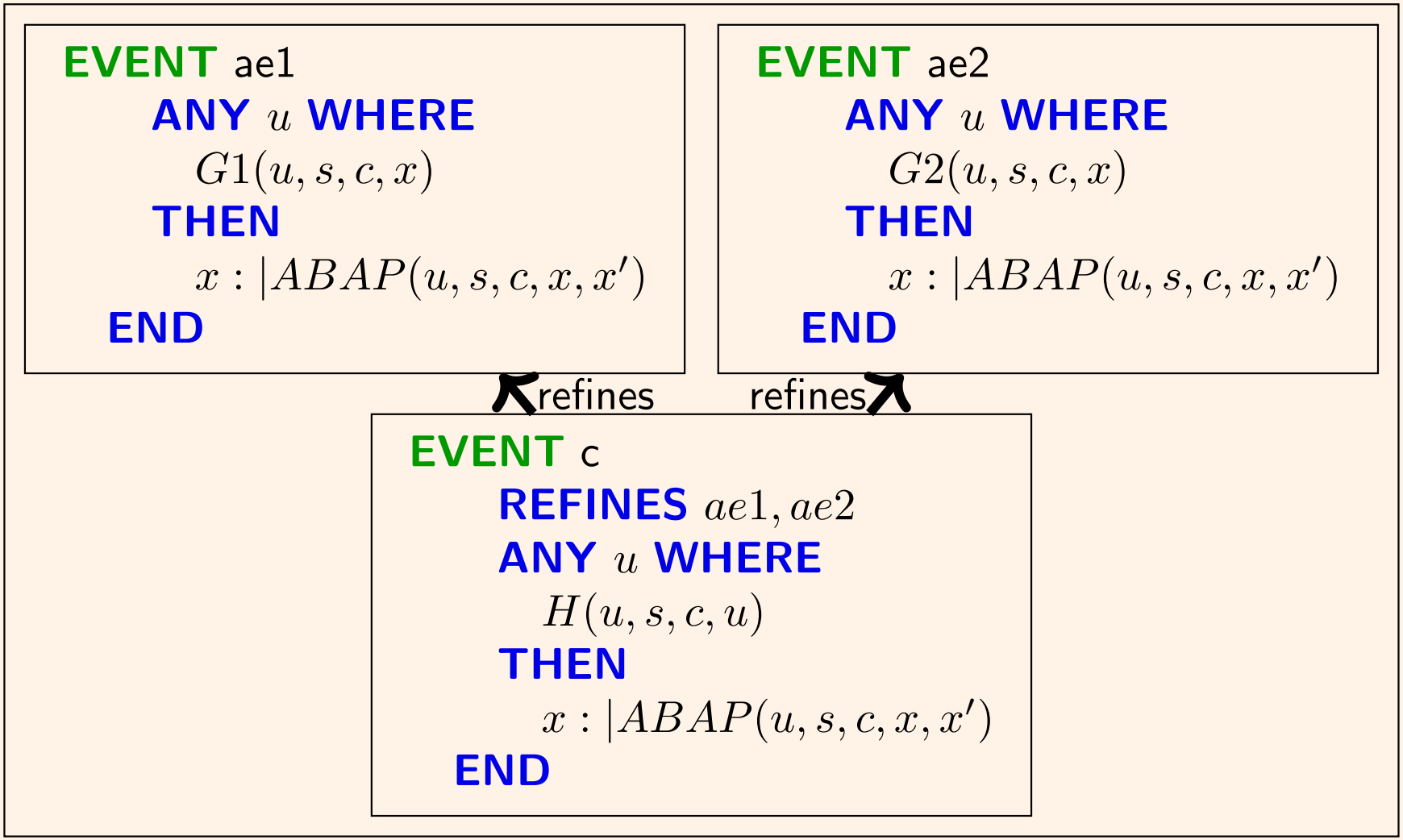
Current Section Summary

1. CM5
2. Overview
3. Programming by contract
   - Short view
   - The call-as-event idea
   - Main idea
4. Strategy 1 : Transformations of the ALGO machine
5. Strategy 2 : Internalizing events by merging
6. Analysis and then Synthesis
   - Analysis using Refinement
   - Synthesis by Merging
7. Summing a list of integer values
8. Stategy 3 : The Recursive Interpretation of call
9. Two examples
   - Computing the square of a number
   - Searching a key in an array

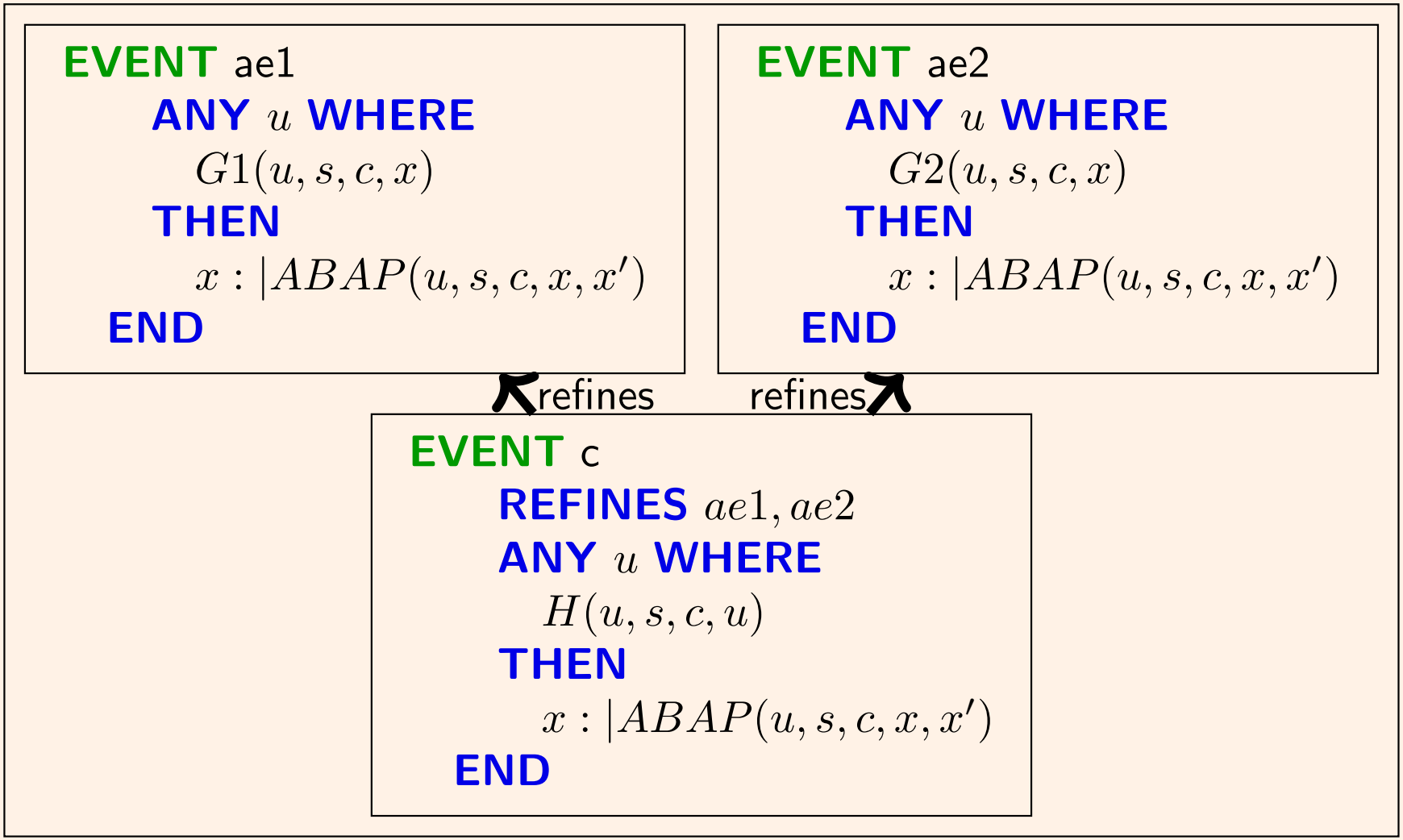
Global Picture



Merging two events



Merging two events



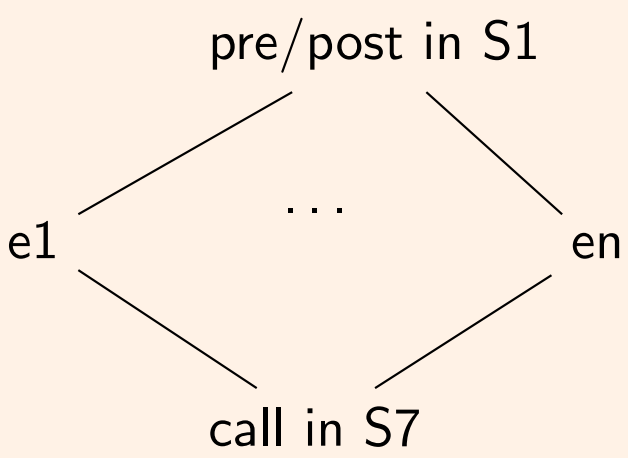
$$AX(s,c), I(s,c,x), H(v,s,c,y) \vdash G_1(s,c,x) \vee G_2(s,c,x).$$

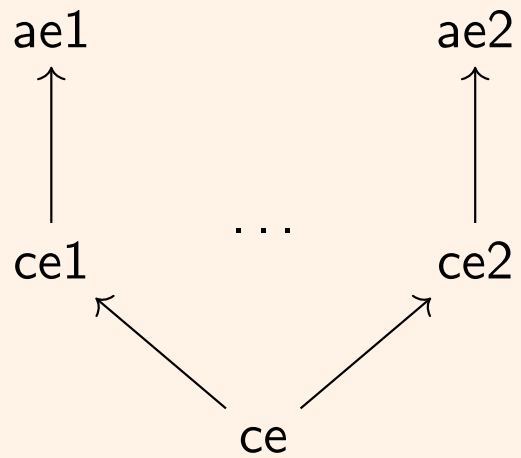
Applying the fusion of events

Strategy for merging events

Let be two events ae1 and ae2 such that the two guards are complementary (ie $g(ae1) \Leftrightarrow \neg g(ae2)$ or $g(ae2) \Leftrightarrow \neg g(ae1)$).

- Let cei be the event with the guard g(aei) and with the action $(g(ae1) \Rightarrow a(ae1)) \wedge (g(ae2 \Rightarrow a(ae2))$.
  Then cei refines aei.
- Let ce be the event with the guard $g(ae1) \vee g(ae2)$.
  Then ce refines cei by fusion.





Current Section Summary

1. CM5
2. Overview
3. Programming by contract
   - Short view
   - The call-as-event idea
   - Main idea
4. Strategy 1 : Transformations of the ALGO machine
5. Strategy 2 : Internalizing events by merging
6. Analysis and then Synthesis
   - Analysis using Refinement
   - Synthesis by Merging
7. Summing a list of integer values
8. Stategy 3 : The Recursive Interpretation of call
9. Two examples
   - Computing the square of a number
   - Searching a key in an array

Current Sub Section Summary

1. CM5
2. Overview
3. Programming by contract
4. Strategy 1 : Transformations of the ALGO machine
5. Strategy 2 : Internalizing events by merging
6. Analysis and then Synthesis
   - Analysis using Refinement
   - Synthesis by Merging
7. Summing a list of integer values
8. Stategy 3 : The Recursive Interpretation of call
9. Two examples

Computing two sums

First n numbers and first odd/even numbers

- First the **pre/post specification** . . .
- Possibility to use a programming language with contracts too

Computing two sums

First n numbers and first odd/even numbers

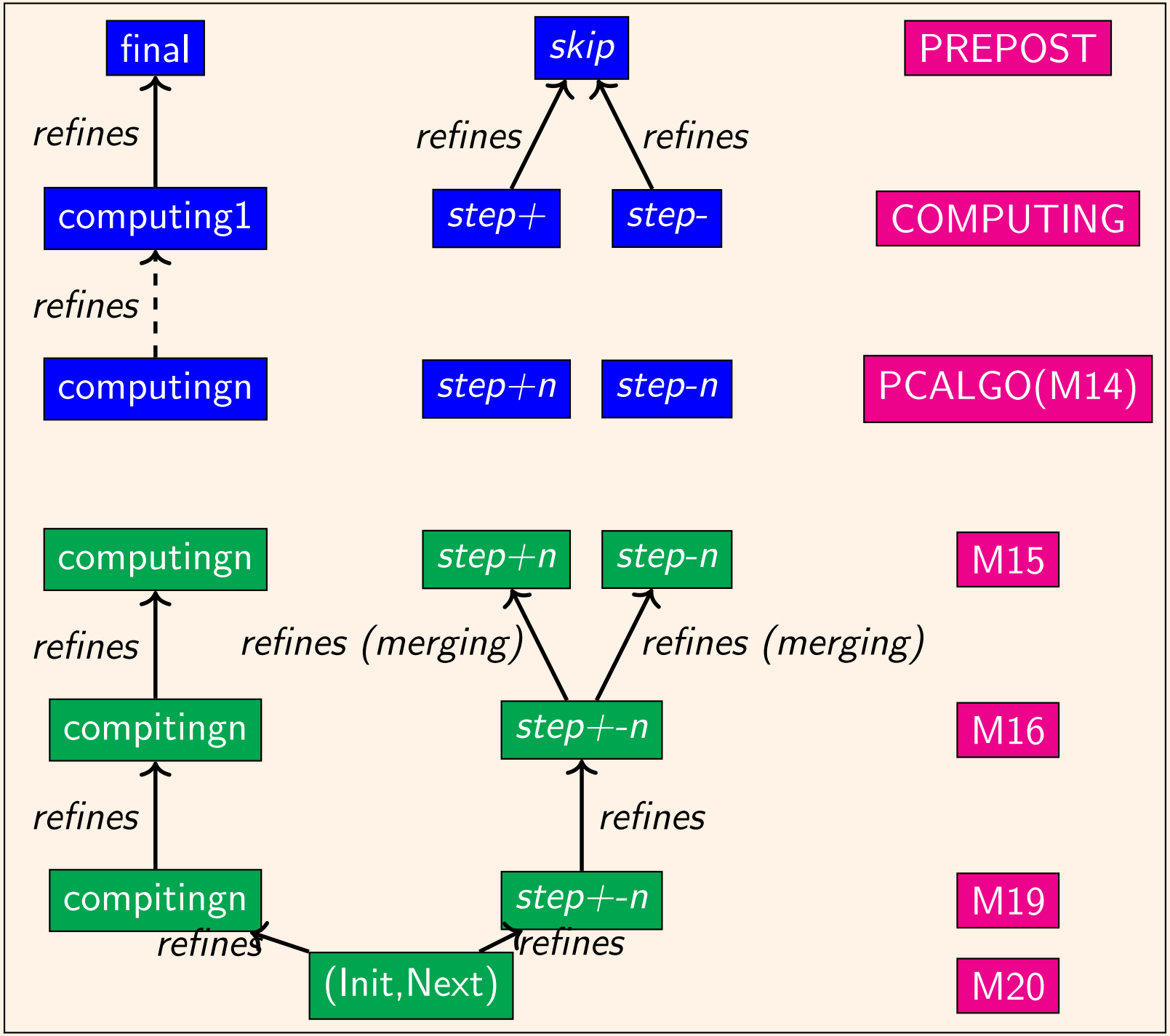
- First the **pre/post specification** ...
- Possibility to use a programming language with contracts too

requires $input_0 \geq 0 \wedge rs_0, re_0 \in \mathbb{Z}$

ensures $\begin{cases} rs_f = s(input_0) \\ re_f = es(input_0) \\ input_f = input_0 \end{cases}$

variables input, re, rs

- Find two sequences s and es computing the sum of first n natural numbers and first even numbers smaller than n
- Prove that :
  - $\forall n.n \in \mathbb{N} \Rightarrow s(n) = \sum_{i=0}^{i=n} i.$
  - $\forall n.n \in \mathbb{N} \Rightarrow es(n) = \sum_{i=0}^{i=n/2} 2 * i.$


final
skip
PREPOST
refines
refines
refines
computing1
step+
step-
COMPUTING
refines
computingn
step+n
step-n
PCALGO(M14)
computingn
step+n
step-n
M15
refines
refines (merging)
refines (merging)
compitingn
step+-n
M16
refines
refines
compitingn
step+-n
M19
refines
refines
(Init,Next)
M20

First n numbers and first odd/even numbers

$axm1 : n \in \mathbb{N}$

$axm2 : s \in \mathbb{N} \to \mathbb{N} \land os \in \mathbb{N} \to \mathbb{N} \land es \in \mathbb{N} \to \mathbb{N}$

$axm3 : es(0) = 0 \land os(0) = 0 \land s(0) = 0$

$axm4 : \forall i, l \cdot i \in \mathbb{N} \land l \in \mathbb{N} \land i = 2 * l \Rightarrow s(i+1) = s(i) + i + 1 \land es(i+1) =$

$axm5 : \forall i, l \cdot i \in \mathbb{N} \land l \in \mathbb{N} \land i = 2 * l + 1 \Rightarrow s(i+1) = s(i) + i + 1 \land es(i +$

$axm6 : suc \in \mathbb{N} \to \mathbb{N} \land (\forall i \cdot i \in \mathbb{N} \Rightarrow suc(i) = i + 1)$

$axm7 : \forall A \cdot A \subseteq \mathbb{N} \land 0 \in A \land suc[A] \subseteq A \Rightarrow \mathbb{N} \subseteq A$

$th1 : \forall i \cdot i \in \mathbb{N} \Rightarrow s(i+1) = s(i) + i + 1$

$th2 : \forall u, v \cdot u \in \mathbb{N} \land v \in \mathbb{N} \land 2 * u = v \Rightarrow u = v/2$

$th3 : \forall k \cdot k \in \mathbb{N} \Rightarrow 2 * s(k) = k * k + k$

$th4 : \forall k \cdot k \in \mathbb{N} \Rightarrow s(k) = (k * k + k)/2$

$th5 : \forall k \cdot k \in \mathbb{N} \Rightarrow es(2 * k) = 2 * s(k)$

$th6 : \forall k \cdot k \in \mathbb{N} \Rightarrow es(2 * k + 1) = 2 * s(k)$

$th7 : \forall k \cdot k \in \mathbb{N} \land k \neq 0 \Rightarrow os(2 * k) = k * k$

$th8 : \forall k \cdot k \in \mathbb{N} \Rightarrow os(2 * k + 1) = (k + 1) * (k + 1)$

Stating the pre/post specification in Event-B

INVARIANTS

$inv1 : input \in \mathbb{Z}$

$inv6 : input = n$

$inv2 : rs \in \mathbb{Z} \land re \in \mathbb{Z}$

$inv3 : ok \in BOOL$

$inv4 : ok = TRUE \Rightarrow rs = s(input) \land re = es(input)$

END

Stating the pre/post specification in Event-B

Refining for computing (1)

INVARIANTS

$inv1 : cur \in 0 \,..\, n$

$inv2 : ee \in 0 \,..\, n \pfun \mathbb{N}$

$inv3 : ss \in 0 \,..\, n \pfun \mathbb{N}$

$inv5 : dom(ss) = 0 \,..\, cur \land dom(ee) = dom(ss) \land dom(ss) \subseteq \mathbb{N}$

$inv6 : \forall i \cdot i \in 0 \,..\, cur \Rightarrow ee(i) = es(i) \land ss(i) = s(i)$

VARIANT

THEN

$act1 : ok := FALSE$

$act2 : rs :\in \mathbb{Z}$

$act3 : re :\in \mathbb{Z}$

$act4 : input := n$

$act5 : ee := \{0 \mapsto 0\}$

$act6 : ss := \{0 \mapsto 0\}$

$act7 : cur := 0$

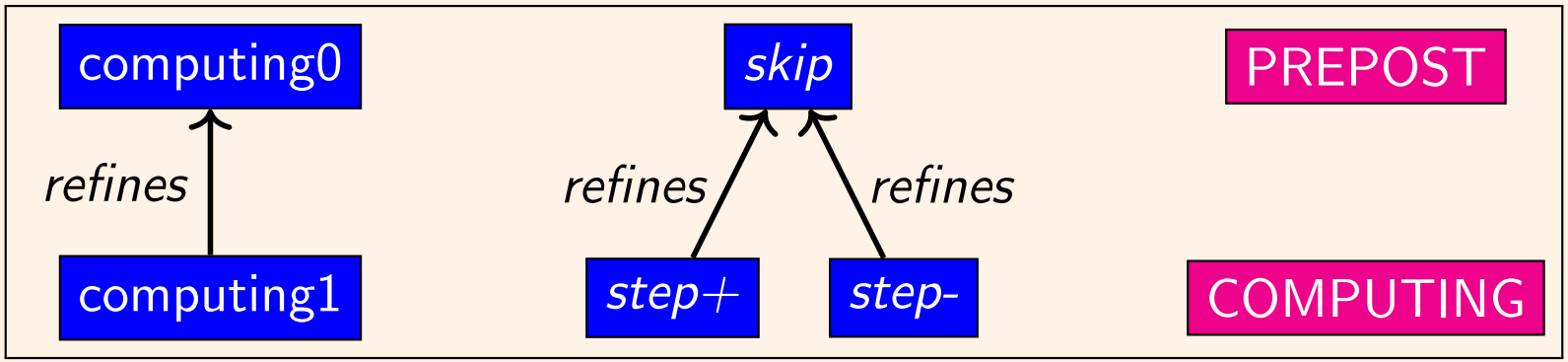
END

Refining for computing (2)

Refining for computing (3)

Refining for computing (4)

Diagram of events



Completing the analysis

MACHINE $M1$ **SEES** $C0$
VARIABLES
$input, rs, re, ok,$
INVARIANTS
$inv1 : input \in \mathbb{Z}$
$inv6 : input = n$
$inv2 : rs \in \mathbb{Z} \wedge re \in \mathbb{Z}$
$inv3 : ok \in BOOL$
$inv4 : ok = TRUE \Rightarrow rs = s(input) \wedge re = es(input)$

Completing the analysis

MACHINE $M11$ **SEES** $C0$
VARIABLES
$input, rs, re, ok, ee, ss, cur,$
INVARIANTS
$inv1 : cur \in 0 \,..\, n$
$inv2 : ee \in 0 \,..\, n \pfun \mathbb{N}$
$inv3 : ss \in 0 \,..\, n \pfun \mathbb{N}$
$inv5 : dom(ss) = 0 \,..\, cur \land dom(ee) = dom(ss) \land dom(ss) \subseteq \mathbb{N}$
$inv6 : \forall i \cdot i \in 0 \,..\, cur \Rightarrow ee(i) = es(i) \land ss(i) = s(i)$

Completing the analysis

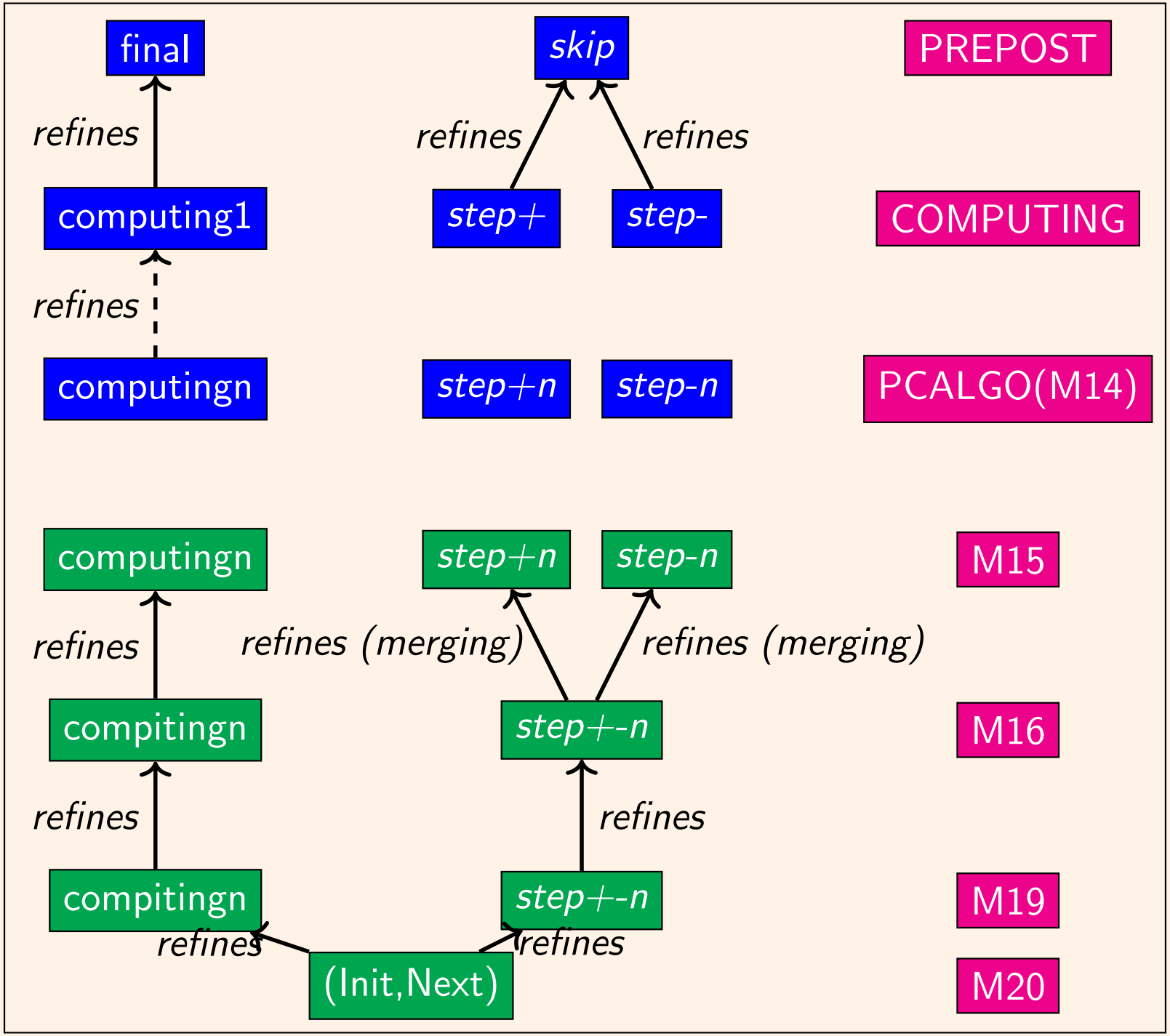
MACHINE $M12$ **SEES** $C0$
VARIABLES
$input, rs, re, ok, ee, ss, cur, cs, ce,$
INVARIANTS
$inv1 : cs = ss(cur)$
$inv2 : ce = ee(cur)$

Completing the analysis

MACHINE $M13$ **SEES** $C0$
VARIABLES
$input, rs, re, ok, ee, ss, cur, cs, ce,$
INVARIANTS
$inv1 : cs = ss(cur)$
$inv2 : ce = ee(cur)$
$inv4 : ce = es(cur)$
$inv3 : cs = s(cur)$

Completing the analysis

MACHINE $M14$ **SEES** $C0, C1$
VARIABLES
$input, rs, re, cur, cs, ce, l,$
INVARIANTS
$inv1 : cs = s(cur)$
$inv5 : l = start \Leftrightarrow ok = FALSE$
$inv6 : l = end \Leftrightarrow ok = TRUE$
$inv2 : cs = s(cur)$
$inv3 : l \in L$
$inv4 : l = end \Rightarrow re = es(n) \wedge rs = s(n)$


final
skip
PREPOST
refines
refines
refines
computing1
step+
step-
COMPUTING
refines
computingn
step+n
step-n
PCALGO(M14)
computingn
step+n
step-n
M15
refines
refines (merging)
refines (merging)
compitingn
step+-n
M16
refines
refines
compitingn
step+-n
M19
refines
refines
(Init,Next)
M20

Generation of an algorithm from M14

Translation from Evenrts

```
struct sums  codesum(int n)
{int  k,ce,cs; struct sums r;
  r.s=0;r.se=0;k=0;ce=0;cs=0;
  while (k<n)
    {
      if ( k % 2 != 0)
        { ce = ce + k + 1;cs = cs +k+1;        k = k +1;}
      else
        { ce = ce ;cs = cs +k+1;        k = k +1;}
       }
  r.s=cs;r.se=ce;
  return(r);
}
```

Generation of an algorithm from M14

Improving and checking with Frama-c

```
#include "structure.h"
#include "trans-even.h"
struct sums  codesum(int n)
{int  k,ce,cs; struct sums r;
  r.s=0;r.se=0;k=0;ce=0;cs=0;
  /*@ loop invariant k >= 0 && k <= n && mathsum(k) == cs;
    @ loop invariant  ( (k % 2 == 0) ==> (mathse(k) == ce));
    @ loop invariant  ( (k % 2 != 0) ==> (mathse(k) == ce));
    loop assigns k, ce,cs;
    loop variant n-k;
   */
  while (k<n)
    {
      if ( k % 2 != 0)
        { ce = ce + k + 1;cs = cs +k+1;        k = k +1;}
      else
        { ce = ce ;cs = cs +k+1;        k = k +1;}
       }
  r.s=cs;r.se=ce;
  return(r);
```

Generation of an algorithm from M14

Improving and checking with Frama-c

```
#include "structure.h"
#include "trans-even.h"
struct sums  codesum(int n)
{int  k,ce,cs; struct sums r;
  r.s=0;r.se=0;k=0;ce=0;cs=0;
  /*@ loop invariant k >= 0 && k <= n && mathsum(k) == cs;
    @ loop invariant  ( (k % 2 == 0) ==> (mathse(k) == ce));
    @ loop invariant  ( (k % 2 != 0) ==> (mathse(k) == ce));
    loop assigns k, ce,cs;
    loop variant n-k;
   */
  while (k<n)
    {
      if ( k % 2 != 0)
        { ce = ce + k + 1;}
      cs = cs +k+1;        k = k +1;
 cp trabs-e*          }
  r.s=cs;r.se=ce;
  return(r);
}
```

Generation of an algorithm from M14

Using the Plugin EB2Algo

```
rs :∈ ℤ ||
re :∈ ℤ ||
input = n ||
cur = 0 ||
ce = 0 ||
cs = 0

while cur≠n do
    if cur mod 2≠0 then
        cur = cur+1 ||
        cs = cs+cur+1 ||
        ce = ce+cur+1
    else
        cur = cur+1 ||
        cs = cs+cur+1 ||
        ce = ce
    od;
```

Current Sub Section Summary

1. CM5
2. Overview
3. Programming by contract
4. Strategy 1 : Transformations of the ALGO machine
5. Strategy 2 : Internalizing events by merging
6. Analysis and then Synthesis
   - Analysis using Refinement
   - Synthesis by Merging
7. Summing a list of integer values
8. Stategy 3 : The Recursive Interpretation of call
9. Two examples

Merging events

EVENT e1
ANY
x
WHERE
$G1(s, c, x, u)$
THEN
$u :\mid (R(s, c, x, u, u'))$
END

EVENT e2
ANY
x
WHERE
$G2(s, c, x, u)$
THEN
$u :\mid (R(s, c, x, u, u'))$
END

Merging events

EVENT e1
ANY
x
WHERE
$G1(s, c, x, u)$
THEN
$u :| (R(s, c, x, u, u'))$
END

EVENT e2
ANY
x
WHERE
$G2(s, c, x, u)$
THEN
$u :| (R(s, c, x, u, u'))$
END

EVENT e $refines$ $e1, e2$
ANY
x
WHERE
$H(s, c, x, u)$
THEN
$u :| (R(s, c, x, u, u'))$
END

Merging events

EVENT e1
ANY
x
WHERE
$G1(s, c, x, u)$
THEN
$u :| (R(s, c, x, u, u'))$
END

EVENT e2
ANY
x
WHERE
$G2(s, c, x, u)$
THEN
$u :| (R(s, c, x, u, u'))$
END

EVENT e $refines$ $e1, e2$
ANY
x
WHERE
$H(s, c, x, u)$
THEN
$u :| (R(s, c, x, u, u'))$
END

Checking the Proof Obligation :

$$Ax(s, c), I(s, c, u), H(s, c, x, u) \vdash G1(s, c, x, u) \vee G2(s, c, x, u)$$

Synthesis Phase

- Preparing merging by transforming actions.

EVENT e
ANY
x
WHERE
$G(s, c, x, u)$
THEN
$u :| (R(s, c, x, u, u'))$
END

EVENT f refines e
ANY
x
WHERE
$G(s, c, x, u)$
THEN
$u :| (G(s, c, x, u) \Rightarrow R(s, c, x, u, u'))$
END

- Internalizing the action as a before after relation
- $G(s, c, x, u) \wedge (G(s, c, x, u) \Rightarrow R(s, c, x, u, u')) \Leftrightarrow$
  $G(s, c, x, u) \wedge R(s, c, x, u, u')$

Synthesis Phase

- Preparing merging by transforming actions.

EVENT e
ANY
x
WHERE
$G(s, c, x, u)$
THEN
$u :| (R(s, c, x, u, u'))$
END

EVENT f refines e
ANY
x
WHERE
$G(s, c, x, u)$
THEN
$u :| (G(s, c, x, u) \Rightarrow R(s, c, x, u, u'))$
END

- $G(s, c, x, u) \wedge (G(s, c, x, u) \Rightarrow R(s, c, x, u, u')) \Leftrightarrow$
$G(s, c, x, u) \wedge R(s, c, x, u, u')$
- $A(s, c, u) \wedge I(s, c, u) \wedge BA(f)(s, c, u, u') \Rightarrow$
$I(s, c, u') \wedge BA(e)(s, c, u, u')$

Synthesis Phase

- Preparing merging by transforming actions.

EVENT e
ANY
x
WHERE
$G(s, c, x, u)$
THEN
$u :| (R(s, c, x, u, u'))$
END

EVENT f refines e
ANY
x
WHERE
$G(s, c, x, u)$
THEN
$u :| (G(s, c, x, u) \Rightarrow R(s, c, x, u, u'))$
END

- $G(s, c, x, u) \wedge (G(s, c, x, u) \Rightarrow R(s, c, x, u, u')) \Leftrightarrow$
  $G(s, c, x, u) \wedge R(s, c, x, u, u')$
- $A(s, c, u) \wedge I(s, c, u) \wedge BA(f)(s, c, u, u') \Rightarrow$
  $I(s, c, u') \wedge BA(e)(s, c, u, u')$
- $A(s, c, u) \wedge I(s, c, u) \wedge G(s, c, x, u) \wedge (G(s, c, x, u) \Rightarrow$
  $R(s, c, x, u, u')) \Rightarrow I(s, c, u') \wedge G(s, c, x, u) \wedge R(s, c, x, u, u')$

Synthesis Phase : review of machines from M15-M20

MACHINE $M15$ **SEES** $C0, C1$
VARIABLES
$input, rs, re, cur, cs, ce, l,$

Synthesis Phase : review of machines from M15-M20

MACHINE $M16$ **SEES** $C0, C1$
VARIABLES
$input, rs, re, cur, cs, ce, l,$

Synthesis Phase : review of machines from M15-M20

MACHINE $M17$ **SEES** $C0, C1$
VARIABLES
$input, rs, re, cur, cs, ce, l,$

Synthesis Phase : review of machines from M15-M20

MACHINE $M18$ **SEES** $C0, C1$
VARIABLES
$input, rs, re, cur, cs, ce, l,$

Synthesis Phase : review of machines from M15-M20

MACHINE $M19$ **SEES** $C0, C1$
VARIABLES
$input, rs, re, cur, cs, ce, l,$

Synthesis Phase : review of machines from M15-M20

MACHINE $M20$ **SEES** $C0, C1$
VARIABLES
$input, rs, re, cur, cs, ce, l,$

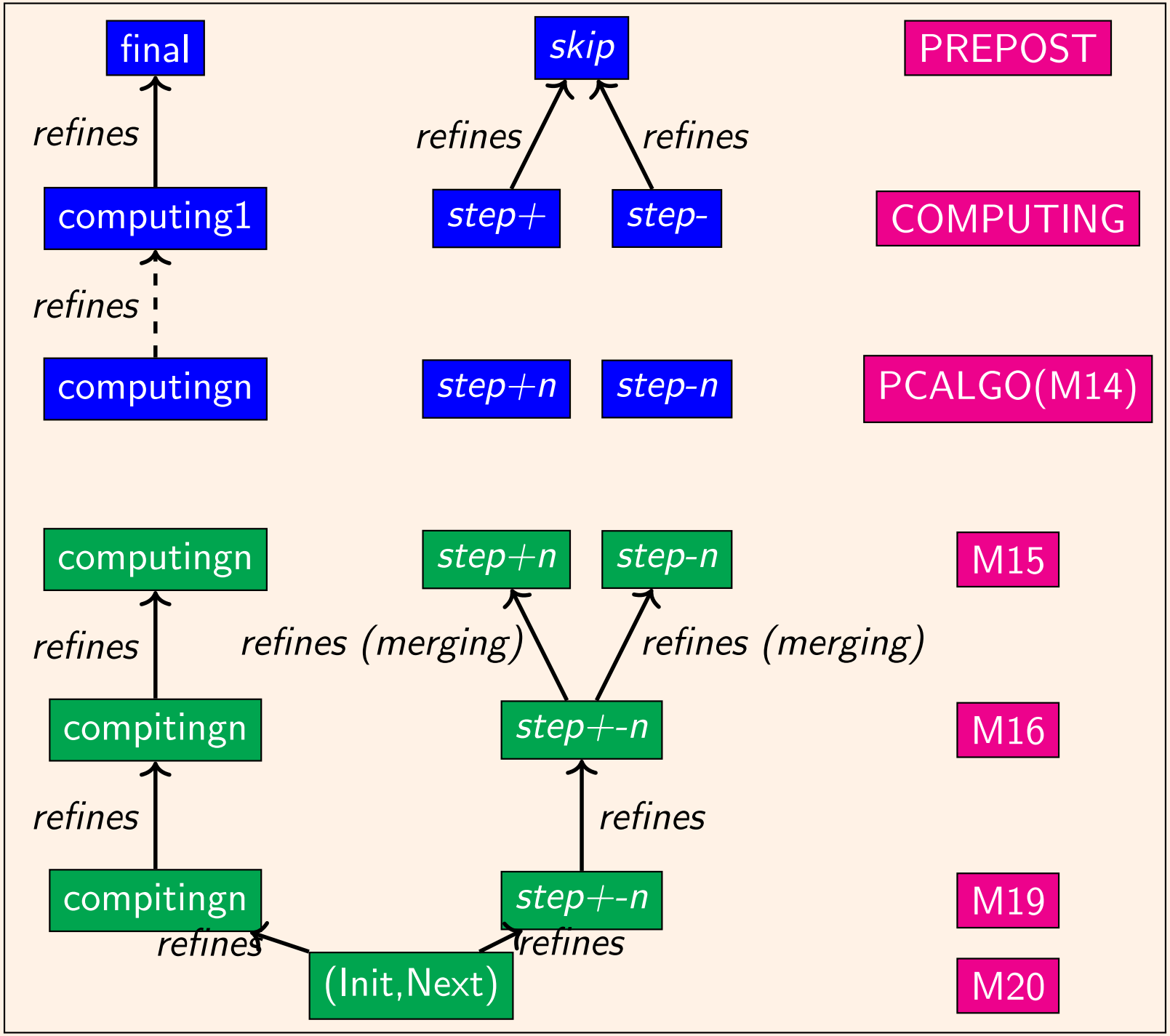
Generating a TLA specification

```
----------------------------
next1 ==
      (l="start" /\   cur < n  /\ ( cur % 2 = 0))
      /\   (cur'=cur+1 /\  ce'=ce /\  l'=l
      /\  cs'=cs+cur+1 /\  re'=re /\  rs'=rs)
o
delsnext2 ==
      (l="start" /\   cur < n  /\ ( cur % 2 # 0))
      /\ (cur'=cur+1 /\  ce'=ce+cur+1 /\  l'=l
      /\  cs'=cs+cur+1 /\  re'=re /\ rs'=rs)

next3 ==
 (l="start" /\  cur=n) /\   (rs'=cs /\  re'=ce
 /\  l'="end" /\  cur'=cur /\ cs'=cs/\ ce'=ce)

Next ==
      (next1 \/ next2 \/ next3)

Init == l="start" /\   cur=0 /\ rs=0 /\ cs=0 /\ re=0 /\ ce =0
=============================================================
```


final
skip
PREPOST
refines
refines
refines
computing1
step+
step-
COMPUTING
refines
computingn
step+n
step-n
PCALGO(M14)
computingn
step+n
step-n
M15
refines
refines (merging)
refines (merging)
compitingn
step+-n
M16
refines
refines
compitingn
step+-n
M19
refines
refines
(Init,Next)
M20

Current Section Summary

1. CM5
2. Overview
3. Programming by contract
   - Short view
   - The call-as-event idea
   - Main idea
4. Strategy 1 : Transformations of the ALGO machine
5. Strategy 2 : Internalizing events by merging
6. Analysis and then Synthesis
   - Analysis using Refinement
   - Synthesis by Merging
7. Summing a list of integer values
8. Stategy 3 : The Recursive Interpretation of call
9. Two examples
   - Computing the square of a number
   - Searching a key in an array

Contract for the problem Summing a list of integer values

Contract for the problem Summing a list of integer values

VARIABLES $n, v, r, done$

REQUIRES
$$\begin{cases} n_0 \in \mathbb{N} \\ n_0 \neq 0 \\ v_0 \in 1..n_0 \rightarrow \mathbb{Z} \\ r_0 \in \mathbb{Z} \\ done_0 \in \mathbb{BOOL} \end{cases}$$

ENSURES
$$\begin{pmatrix} r_f = \sum_{k=1}^{n_0} v_0(k) \\ done_f = TRUE \end{pmatrix}$$

Contract for the problem Summing a list of integer values

Contract for the problem Summing a list of integer values

VARIABLES $n, v, r, done$

REQUIRES
$$\begin{cases} n_0 \in \mathbb{N} \\ n_0 \neq 0 \\ v_0 \in 1..n_0 \rightarrow \mathbb{Z} \\ r_0 \in \mathbb{Z} \\ done_0 \in \mathbb{BOOL} \end{cases}$$

ENSURES
$$\begin{pmatrix} r_f = \sum_{k=1}^{n_0} v_0(k) \\ done_f = TRUE \end{pmatrix}$$

- the initial values of n, v and done, denoted n_0, v_0, $done_0$ and the final value of r denoted r_f.
- the final value of the variable r denoted r_f is the sum $\sum_{k=1}^{n_0} v_0(k)$.

Contract for the problem Summing a list of integer values

Contract for the problem Summing a list of integer values

VARIABLES $n, v, r, done$

REQUIRES
$$\begin{cases} n_0 \in \mathbb{N} \\ n_0 \neq 0 \\ v_0 \in 1..n_0 \rightarrow \mathbb{Z} \\ r_0 \in \mathbb{Z} \\ done_0 \in \mathbb{BOOL} \end{cases}$$

ENSURES
$$\begin{pmatrix} r_f = \sum_{k=1}^{n_0} v_0(k) \\ done_f = TRUE \end{pmatrix}$$

- the initial values of n, v and done, denoted n_0, v_0, $done_0$ and the final value of r denoted r_f.
- the final value of the variable r denoted r_f is the sum $\sum_{k=1}^{n_0} v_0(k)$.

computation method

Defining the computation process by defining a sequence u in the context S0

computation method = inductive process

CONTEXT $S0$
CONSTANTS $n0\ v0\ u$
AXIOMS
$@pre1\ n0 \in \mathbb{N}$
$@pre2\ n0 \neq 0$
$@pre5\ v0 \in 1..n0 \to \mathbb{Z}$
$@axm1\ u \in 0..n0 \to \mathbb{Z}$
$@axm2\ u(0) = 0$
$@axm3\ \forall k.\ k \in 1..n0$
$\Rightarrow\ u(k) = u(k-1) + v0(k)$
end

computation method = inductive process

CONTEXT $S0$
CONSTANTS $n0\ v0\ u$
AXIOMS
$@pre1\ n0 \in \mathbb{N}$
$@pre2\ n0 \neq 0$
$@pre5\ v0 \in 1..n0 \to \mathbb{Z}$
$@axm1\ u \in 0..n0 \to \mathbb{Z}$
$@axm2\ u(0) = 0$
$@axm3\ \forall k.\ k \in 1..n0 \Rightarrow u(k) = u(k-1) + v0(k)$
end

- The context S0 defines the sequence u supporting the computation method by definition of the summation expression.

computation method = inductive process

CONTEXT $S0$
CONSTANTS $n0\ v0\ u$
AXIOMS
$@pre1\ n0 \in \mathbb{N}$
$@pre2\ n0 \neq 0$
$@pre5\ v0 \in 1..n0 \to \mathbb{Z}$
$@axm1\ u \in 0..n0 \to \mathbb{Z}$
$@axm2\ u(0) = 0$
$@axm3\ \forall k.\ k \in 1..n0 \Rightarrow u(k) = u(k-1) + v0(k)$
end

- The context S0 defines the sequence u supporting the computation method by definition of the summation expression.
- The refinement process starts by the specification namely S1 and leads to the machine S7 which is containing the event final internalizing the computation of the sum.

Expressing pre/post in a machine S1 (I)

MACHINE $S1$ SEES $S0$
VARIABLES $r\ v\ n\ done$
INVARIANTS
$@inv1\ r \in \mathbb{Z}$
$@inv2\ n \in \mathbb{Z}$
$@inv3\ v \in 1..n \rightarrow \mathbb{Z}$
$@inv4\ done \in BOOL$
$@read-values\ n = n0 \wedge v = v0$
$@post\ done = TRUE \Rightarrow r = u(n0)$
EVENT $INITIALISATION$
then
$@act1\ r :\in \mathbb{Z}$
$@act2\ n := n0$
$@act3\ v := v0$
$@act4\ done := FALSE$
end

- v and n are variables getting the input values as $v0$ and $n0$: *read-values* lebel.
- r is getting the result of the computing process under the control of *done* : post label.
- *post* : expression of the partial correctness.

Expressing pre/post in a machine S1 (II)

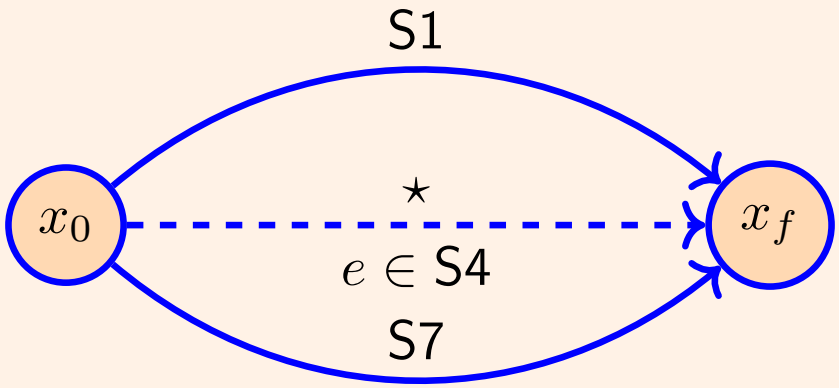
- The final1 event expresses the relationship between intial and final values of variables.
- The keep event expresses the *hidden* loop preparing the final resylt.

EVENT $final1$
  where
  $@grd1\ done = FALSE$
  then
  $@act1\ r := u(n)$
  $@act2\ done := TRUE$
end

$anticipated$ EVENT $keep$
  then
  $@act1\ r, n, v, done :|$
  $(r' \in \mathbb{Z} \land n' \in \mathbb{Z}$
  $\land\ v' \in 1..n' \to \mathbb{Z}$
  $\land\ done' \in BOOL$
  $\land\ n' = n0 \land v' = v0$
  $\land\ (done' = TRUE$
  $\Rightarrow\ r' = u(n0)))$
end

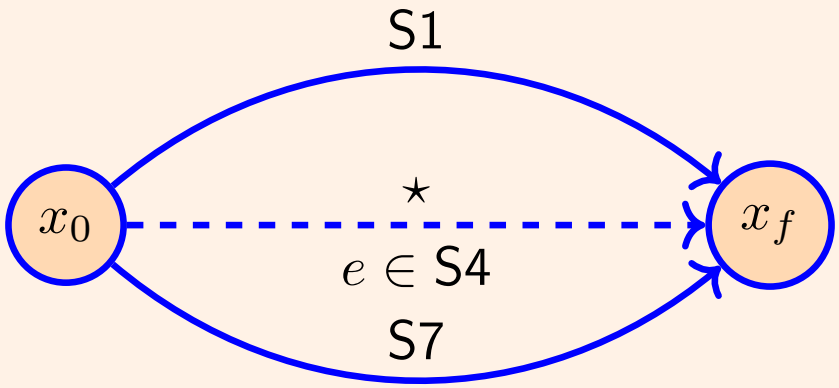
Summary of the refinement steps

Refinement from S1 till S7.



Summary of the refinement steps

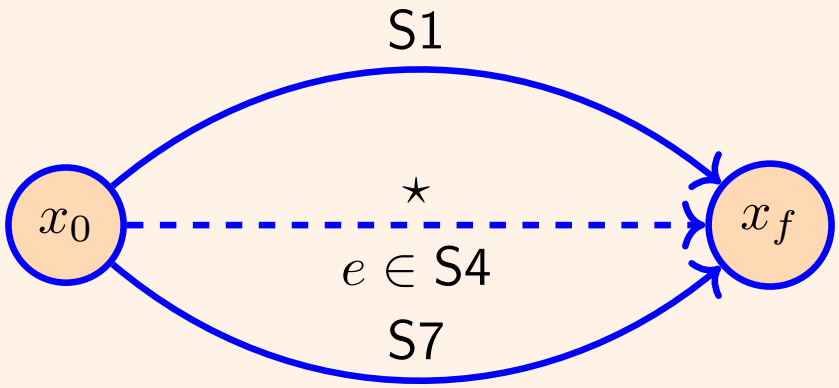
Refinement from S1
till S7.



- Identifying the final1 event as simulating the pre/post contract in S1.

Summary of the refinement steps

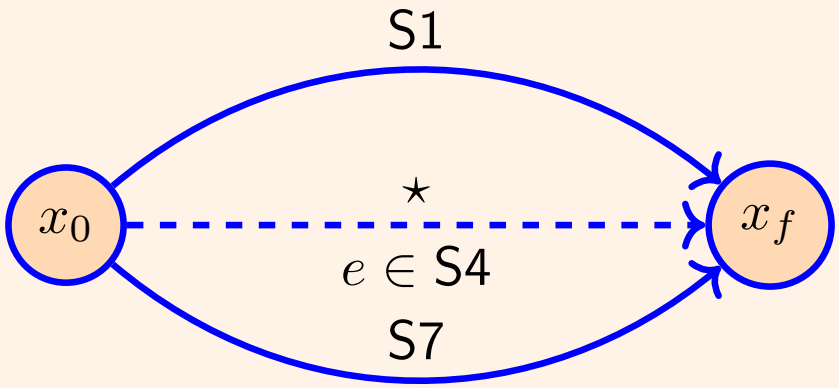
Refinement from S1
till S7.



- Identifying the final1 event as simulating the pre/post contract in S1.
- Refining by using u and introducing events simulating computing steps in S4.

Summary of the refinement steps

Refinement from S1
till S7.



- Identifying the final1 event as simulating the pre/post contract in S1.
- Refining by using u and introducing events simulating computing steps in S4.
- Merging events of S4 into one event in S7.

From S1 to S4 via S2 and S3

From S1 to S4 via S2 and S3

MACHINE $S2$ REFINES $S1$ SEES $S0$
VARIABLES $r\ i\ uu\ n\ v\ done$
INVARIANTS
$@inv1\ i \in 0..n$
$@inv2\ uu \in 0..n \pfun \mathbb{Z} \land dom(uu) = 0..i$
$@inv3\ \forall k.\ k \in dom(uu) \Rightarrow uu(k) = u(k)$
$@inv4\ done = FALSE \Rightarrow i \in 0..n$
$@inv6\ done = TRUE \Rightarrow i = n$
$@inv7\ i \in 0..n-1 \Rightarrow done = FALSE$
$theorem\ @datath\ v = v0 \land n = n0$
VARIANT $n - i$

- The variable uu is recording the sequence u by computing until i is n0.
- The new event step is refining the event keep and decreases the variant.

From S1 to S4 via S2 and S3

MACHINE $S2$ REFINES $S1$ SEES $S0$
VARIABLES $r\ i\ uu\ n\ v\ done$
INVARIANTS
$@inv1\ i \in 0..n$
$@inv2\ uu \in 0..n \pfun \mathbb{Z} \land dom(uu) = 0..i$
$@inv3\ \forall k.\ k \in dom(uu) \Rightarrow uu(k) = u(k)$
$@inv4\ done = FALSE \Rightarrow i \in 0..n$
$@inv6\ done = TRUE \Rightarrow i = n$
$@inv7\ i \in 0..n-1 \Rightarrow done = FALSE$
$theorem\ @datath\ v = v0 \land n = n0$
VARIANT $n - i$

MACHINE $S3$ REFINES $S2$ SEES $S0$
VARIABLES $r\ i\ uu\ cu\ n\ v\ done$
INVARIANTS
$@inv1\ cu \in \mathbb{Z}$
$@inv2\ cu = u(i)$

- The variable uu is recording the sequence u by computing until i is n0.
- The new event step is refining the event keep and decreases the variant.
- The new variable cu is recoring the last computed valuye of uu.
- Proofs of the invariants are made easier.

The machine S4 in the middle (I)

MACHINE $S4$ REFINES $S3$ SEES $S0$
VARIABLES $r\ i\ cu\ n\ v\ done$
INVARIANTS
$theorem\ @inv1\ cu = u(i)$
EVENTS
EVENT $INITIALISATION$
then
$@act1\ r :\in \mathbb{Z}$
$@act2\ i := 0$
$@act4\ cu := 0$
$@act5\ v := v0$
$@act6\ n := n0$
$@act7\ done := FALSE$
end

The machine S4 in the middle (I)

MACHINE $S4$ REFINES $S3$ SEES $S0$
VARIABLES $r\ i\ cu\ n\ v\ done$
INVARIANTS
$theorem\ @inv1\ cu = u(i)$
EVENTS
EVENT $INITIALISATION$
then
$@act1\ r :\in \mathbb{Z}$
$@act2\ i := 0$
$@act4\ cu := 0$
$@act5\ v := v0$
$@act6\ n := n0$
$@act7\ done := FALSE$
end

- The variable uu helps in the proof process but is expensive.
- The variable uu is hidden in the abstractions.

The machine S4 in the middle (II)

EVENT $final$ REFINES $final$
where
$@grd1\ i = n$
$@grd2\ done = FALSE$
then
$@act1\ r := cu$
$@act2\ done := TRUE$
end

EVENT $step000$ REFINES $step$
where
$@grd1\ i < n$
$@grd2\ done = FALSE$
then
$@act1\ i := i + 1$
$@act3\ cu := cu + v(i + 1)$
end

The machine S4 in the middle (II)

EVENT $final$ REFINES $final$
where
$@grd1\ i = n$
$@grd2\ done = FALSE$
then
$@act1\ r := cu$
$@act2\ done := TRUE$
end

EVENT $step000$ REFINES $step$
where
$@grd1\ i < n$
$@grd2\ done = FALSE$
then
$@act1\ i := i + 1$
$@act3\ cu := cu + v(i + 1)$
end

- Events are made simpler and uu is hidden.
- Events are guarded by evaluable conditions as $i < n$ which is implying $i \notin dom(uu)$
- Using EB2ALGO, one gets an algorithm but no proof is given of the transformations.

The machine S7 with one event

MACHINE $S7$ REFINES $S6$ SEES $S0$
VARIABLES $r\ i\ cu\ n\ v\ done$
INVARIANTS
$theorem\ @inv1\ done = FALSE \Rightarrow i \in 0..n$
$theorem\ @inv2\ done = TRUE \Rightarrow i = n$
$theorem\ @inv3\ i \in 0..n-1 \Rightarrow done = FALSE$
$theorem\ @inv4\ i \in 0..n$
EVENT $final$ REFINES $final$
where
$@grd2\ done = FALSE$
then
$@act1\ r, i, cu, n, v, done :| ($
$(i = n$
$\Rightarrow (r' = cu \wedge cu' = cu \wedge i' = i \wedge n' = n$
$\wedge\ v' = v \wedge done' = TRUE))$
$\wedge\ (i < n$
$\Rightarrow (\ r' = r \wedge cu' = cu + v(i+1) \wedge i' = i+1$
$\wedge\ n' = n \wedge v' = v \wedge done' = done))$
$)$
end

Defining an algorithm by a set of actions

Defining an algorithm by a set of actions

$$guard(r, i, cu, n, v, done) \equiv done = FALSE$$

$$action \equiv \left[\begin{array}{ll} i = n \Rightarrow & \left[\begin{array}{l} (r' = cu \wedge cu' = cu \wedge i' = i \wedge n' = n \\ \wedge v' = v \wedge done' = TRUE) \end{array} \right. \\ i < n \Rightarrow & \left[\begin{array}{l} r' = r \wedge cu' = cu + v(i+1) \wedge i' = i + 1 \wedge n' = n \\ \wedge v' = v \wedge done' = done \end{array} \right. \end{array} \right.$$

Defining an algorithm by a set of actions

$$guard(r,i,cu,n,v,done) \equiv done = FALSE$$

$$action \equiv \left[\begin{array}{ll} i = n \Rightarrow & \left[\begin{array}{l} (r' = cu \wedge cu' = cu \wedge i' = i \wedge n' = n \\ \wedge v' = v \wedge done' = TRUE) \end{array}\right. \\ i < n \Rightarrow & \left[\begin{array}{l} r' = r \wedge cu' = cu + v(i+1) \wedge i' = i+1 \wedge n' = n \\ \wedge v' = v \wedge done' = done \end{array}\right. \end{array}\right.$$

- The machine S7 contains an event Initialisation and a next relation
-

Defining an algorithm by a set of actions

$$
\begin{array}{l}
guard(r,i,cu,n,v,done) \equiv done = FALSE \\
action \equiv \left[\begin{array}{ll}
i = n \Rightarrow & \left[\begin{array}{l} (r' = cu \wedge cu' = cu \wedge i' = i \wedge n' = n \\ \wedge v' = v \wedge done' = TRUE) \end{array} \right. \\
i < n \Rightarrow & \left[\begin{array}{l} r' = r \wedge cu' = cu + v(i+1) \wedge i' = i+1 \wedge n' = n \\ \wedge v' = v \wedge done' = done \end{array} \right.
\end{array} \right.
\end{array}
$$

- The machine S7 contains an event Initialisation and a next relation
-

Definition

(mapping of a binary relation)
A binary relation R on $State$ can be mapped to a GKAT term e for the interpretation i, if $\mathcal{R}_i(e) = R$ and we write $map_i(R) = e$

Defining an algorithm by a set of actions (I)

Guarded Kleene Algebra with Tests

Summing a list of integer values is now considering the iteration property and we obtain

$$init.a^{(c)}b$$

where

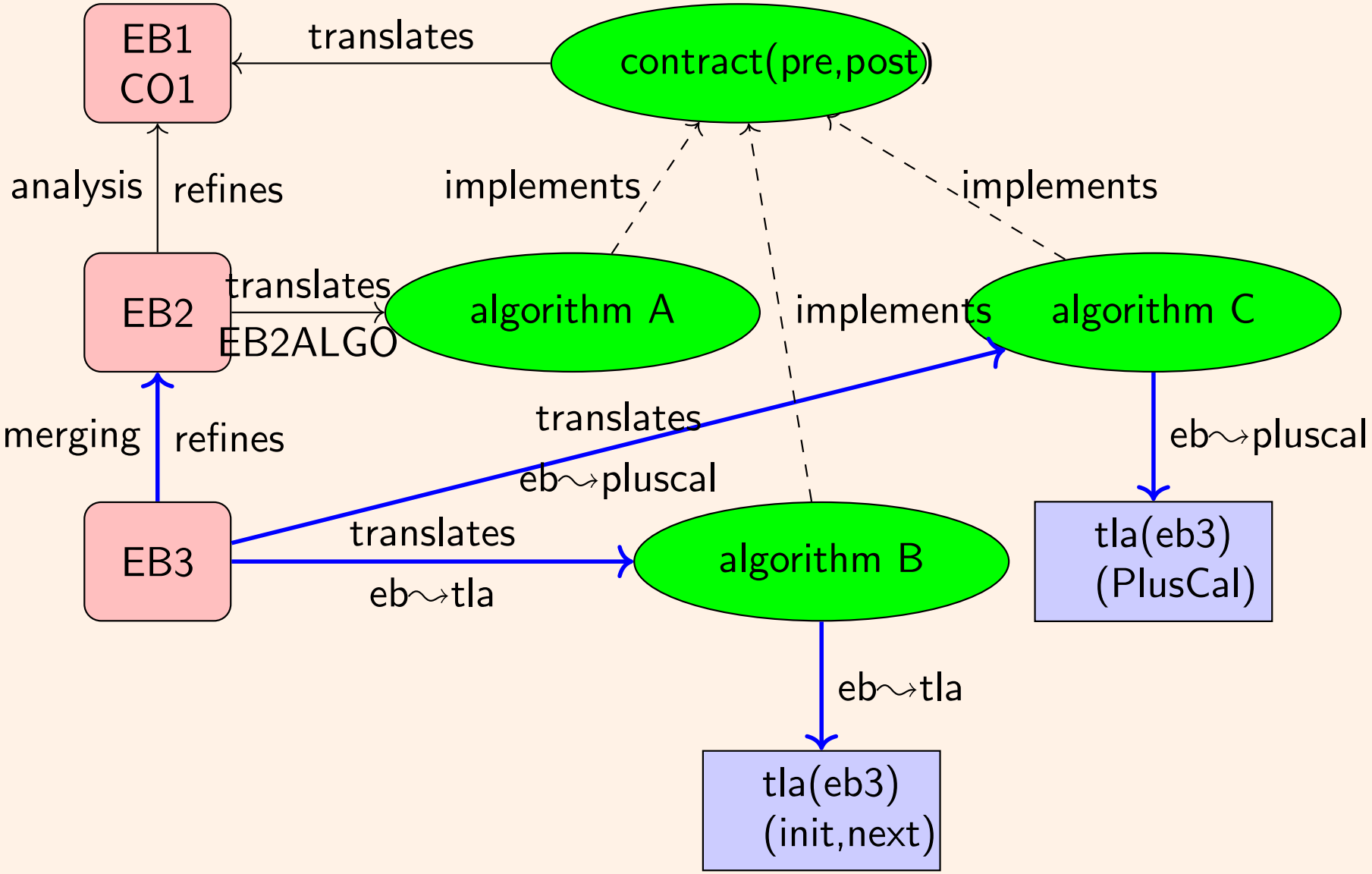
- $sat(init) = \{(n0, v0, FALSE, 0, 0, ?)\}$
- $\mathcal{R}_i(A)(s, s') \triangleq cu' = cu + v(i+1) \wedge i' = i+1 \wedge r' = r$, $a \triangleq cu := cu + v(i+1); i := i+1$ and $map_i(A) = a$
- $\mathcal{R}_i(B)(s, s') \triangleq r' = cu \wedge done' = TRUE \wedge cu' = cu \wedge i' = i$ and $b \triangleq r := cu; done := TRUE$
- $eval(c) = \{(s, s) | s(i) < s(n)\}$.

Defining an algorithm by a set of actions (II)

Listing 4 – PlusCal algorithm for summing a vector

```
  --algorithm sumvect {
variables n=n0, v=v0, i=0, cu=0, r;
{
w1: while (i # n) {
   w2: cu:=cu+v[i+1]; i:=i+1;
};
w3: r:=cu; Gu
}}
```

Global View


EB1
CO1
translates
contract(pre,post)
analysis
refines
implements
implements
EB2
translates
EB2ALGO
algorithm A
implements
algorithm C
merging
refines
translates
eb⇝pluscal
eb⇝pluscal
EB3
translates
eb⇝tla
algorithm B
tla(eb3)
(PlusCal)
eb⇝tla
tla(eb3)
(init,next)

Global View

- Algorithm A is generated using EB2Algo
-

$$
\begin{array}{l}
r :\in \mathbb{Z} \parallel i := 0 \parallel cu := 0 \parallel v := v0 \parallel n := n0 \\
while\ i < n\ do \\
\quad i := i + 1 \parallel cu := cu + v(i+1) \\
od;
\end{array}
$$

Global View

- Algorithm B is simply defined by

$$guard(r,i,cu,n,v,done) \equiv done = FALSE$$

$$action \equiv \left[\begin{array}{ll} i = n \Rightarrow & \left[\begin{array}{l} (r' = cu \wedge cu' = cu \wedge i' = i \wedge n' = n \\ \wedge v' = v \wedge done' = TRUE) \end{array} \right. \\ i < n \Rightarrow & \left[\begin{array}{l} r' = r \wedge cu' = cu + v(i+1) \wedge i' = i+1 \wedge n' = n \\ \wedge v' = v \wedge done' = done \end{array} \right. \end{array} \right.$$

Global View

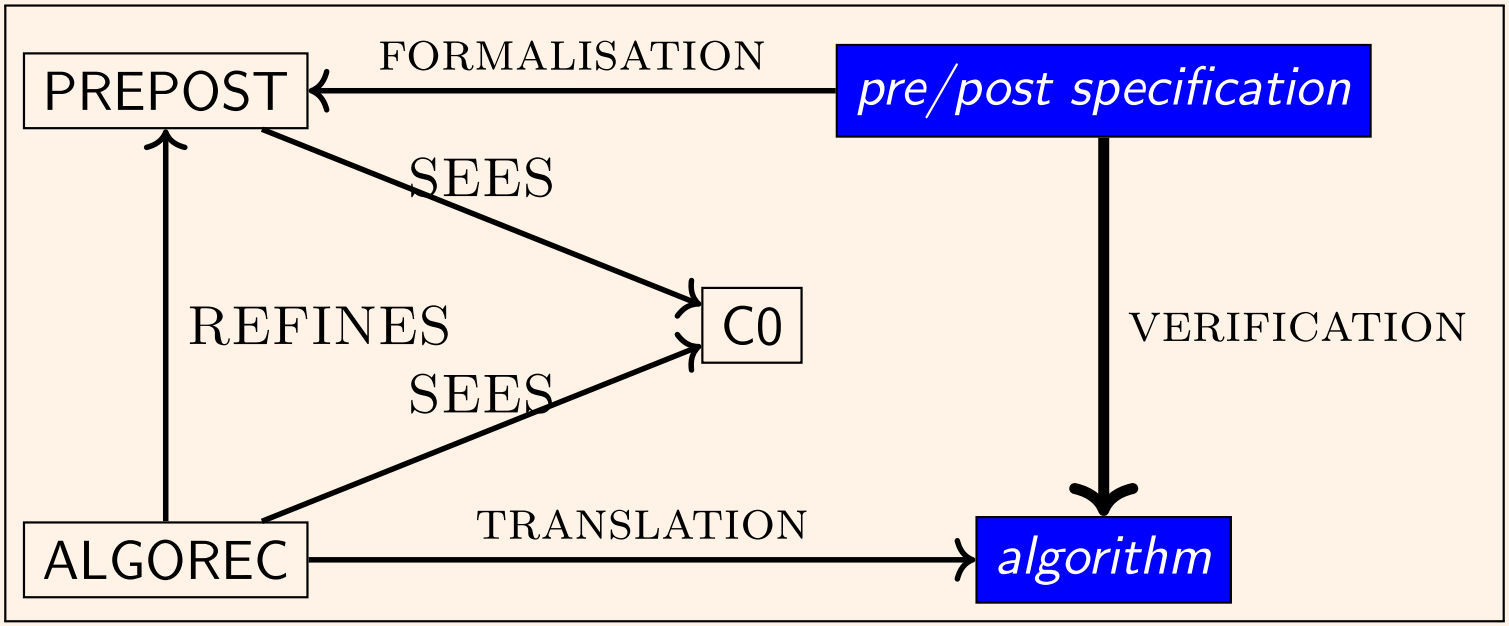
Algorithm C is generated using our transformations based on GKAT algebra

```
      --algorithm sumvect  {
variables  n=n0,v=v0,i=0,cu=0,r;
{
w1:while (i # n) {
  w2:cu:=cu+v[i+1]; i:=i+1;
};
w3:r:=cu;
}}
```

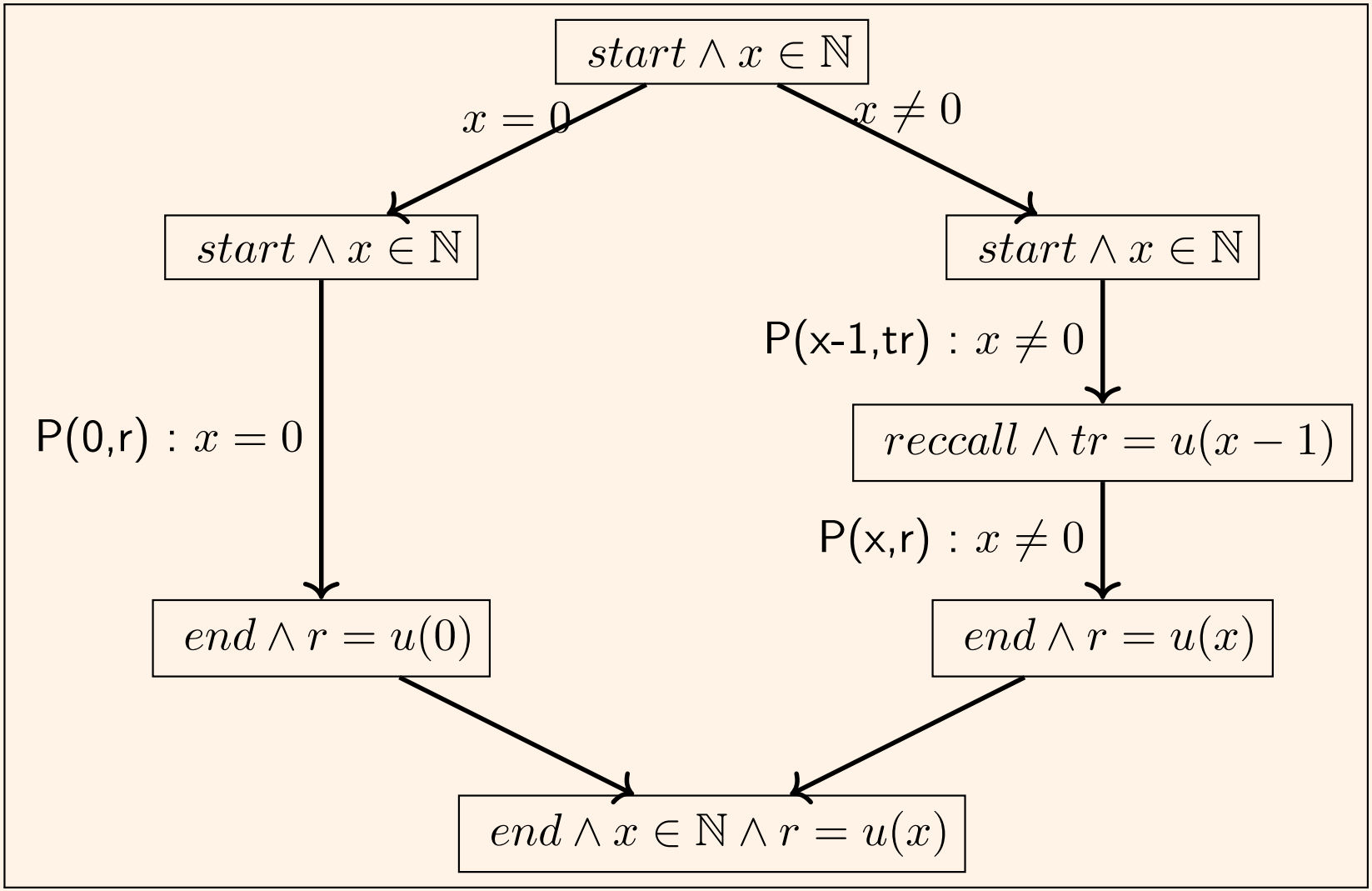
Current Section Summary

1. CM5
2. Overview
3. Programming by contract
   - Short view
   - The call-as-event idea
   - Main idea
4. Strategy 1 : Transformations of the ALGO machine
5. Strategy 2 : Internalizing events by merging
6. Analysis and then Synthesis
   - Analysis using Refinement
   - Synthesis by Merging
7. Summing a list of integer values
8. Stategy 3 : The Recursive Interpretation of call
9. Two examples
   - Computing the square of a number
   - Searching a key in an array

The recursive pattern



The refinement diagram



Ideas for producing algorithms

- **EVENT** rec%PROC(h(x),y)%P(y) is simply simulating the recursive call of the same function.
- The invariant is defined in a simpler way by analysing the inductive structure and a control variable is introduced for structuring the inductive computation.

EVENT
e
WHERE
$\ell = \ell_1$
$g_{\ell_1,\ell_2}(x)$
THEN
$\ell := \ell_2$
$x := f_{\ell_1,\ell_2}(x)$
END

EVENT
rec%PROC(h(x),y)%P(y)
ANY y
WHERE
$\ell = \ell_1$
$g_{\ell_1,\ell_2}(x, y)$
THEN
$\ell := \ell_2$
$x := f_{\ell_1,\ell_2}(x, y)$
END

EVENT
call%APROC(h(x),y)%P(y)
ANY y
WHERE
$\ell = \ell_1$
$g_{\ell_1,\ell_2}(x, y)$
THEN
$\ell := \ell_2$
$x := f_{\ell_1,\ell_2}(x, y)$
END

The computation of the function x^2

VARIABLES

r, c, tr

INVARIANTS

$inv1 : r \in \mathbb{N}$

$inv2 : c = end \Rightarrow r = n * n$

$inv3 : c = callrec \Rightarrow n \neq 0$

$inv4 : c = callrec \Rightarrow tr = (n-1) * (n-1)$

$inv5 : c \in C$

$inv6 : tr \in \mathbb{N}$

$inv7 : c = end \Rightarrow r = n * n$

$inv8 : c = end \wedge n \neq 0$
$\Rightarrow tr = (n-1) * (n-1) \wedge r = tr + 2 * (n-1) + 1$

$inv9 : c = callrec \Rightarrow n * n = tr + 2 * (n-1) + 1$

The computation of the function x^2

EVENT INITIALISATION
BEGIN
$act1 : r := 0$
$act2 : c := start$
$act3 : tr :\in \mathbb{N}$
END
EVENT square0
REFINES square(n ;r)
WHEN
$grd1 : c = start$
$grd2 : n = 0$
THEN
$act1 : c := end$
$act2 : r := 0$
END

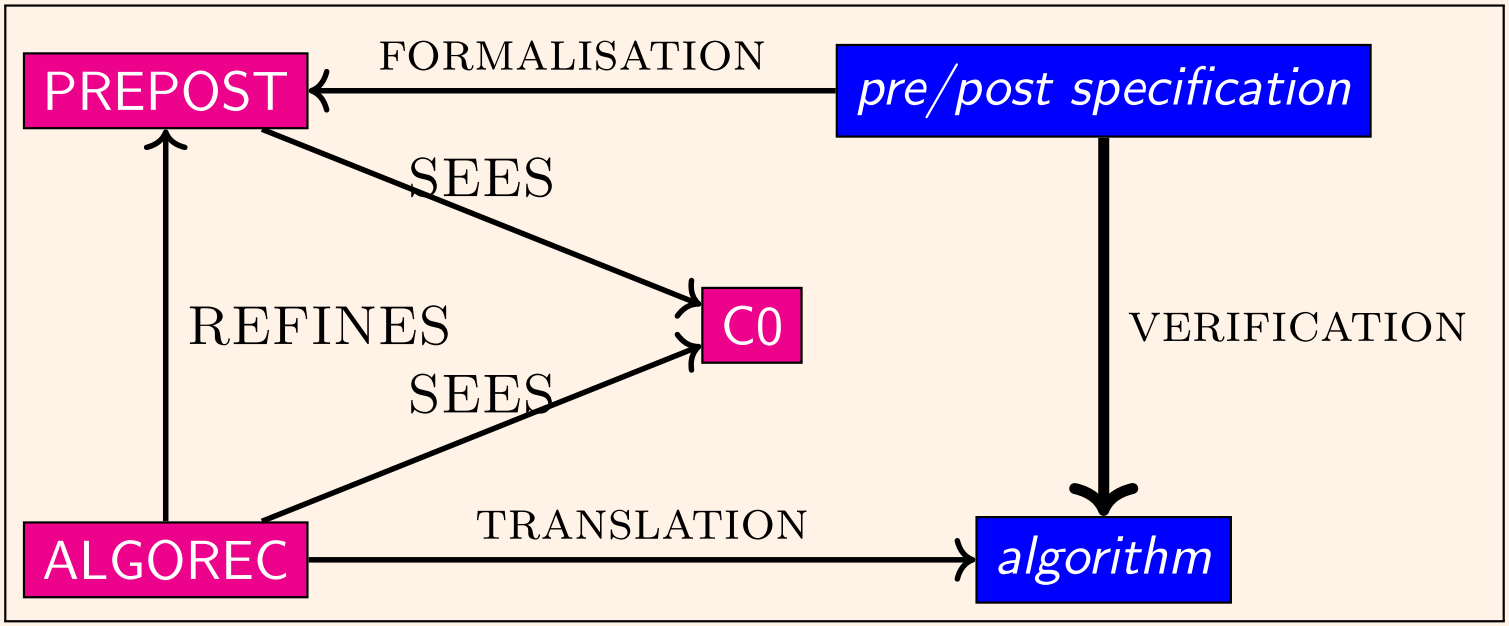
EVENT squaren
REFINES square(n ;r)
WHEN
$grd1 : c = callrec$
THEN
$act1 : r := tr + 2 * (n - 1) + 1$
$act2 : c := end$
END
EVENT rec%square(n-1 ;tr)
WHEN
$grd1 : c = start$
$grd2 : n \neq 0$
THEN
$act1 : c := callrec$
$act2 : tr := (n - 1) * (n - 1)$
END

Summary for proof obligations

Summary for proof obligations

Name	Total	Automatic	Interactive
cae-square	34	32	2
square0	3	2	1
specquare	2	2	0
square	29	28	1

The recursive pattern



Comments

- Proofs are easier and simpler
- Invariant is simple to find
- Translation is automatic
- Students are not happy with it and tools are not always set for this verification.

Current Section Summary

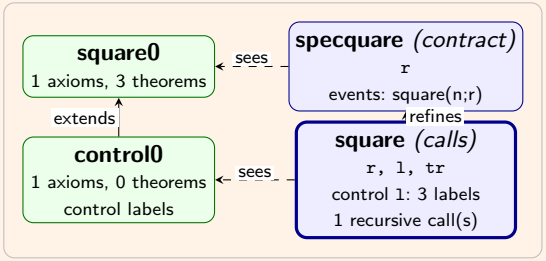
1. CM5
2. Overview
3. Programming by contract
   - Short view
   - The call-as-event idea
   - Main idea
4. Strategy 1 : Transformations of the ALGO machine
5. Strategy 2 : Internalizing events by merging
6. Analysis and then Synthesis
   - Analysis using Refinement
   - Synthesis by Merging
7. Summing a list of integer values
8. Stategy 3 : The Recursive Interpretation of call
9. Two examples
   - Computing the square of a number
   - Searching a key in an array

Current Sub Section Summary

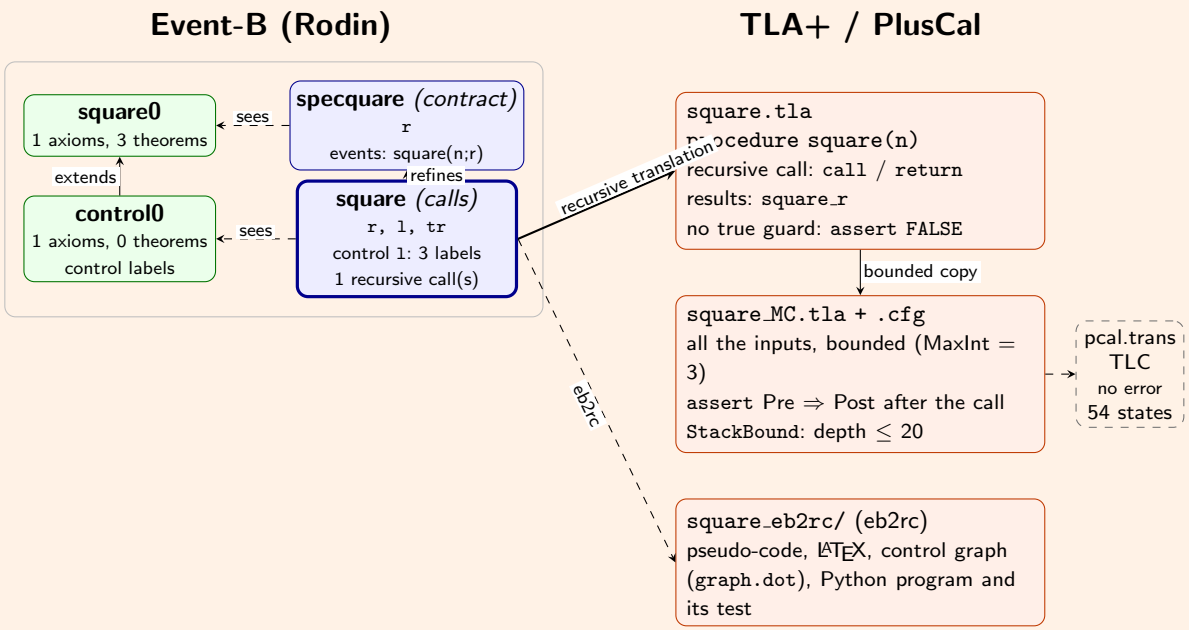
1. CM5
2. Overview
3. Programming by contract
4. Strategy 1 : Transformations of the ALGO machine
5. Strategy 2 : Internalizing events by merging
6. Analysis and then Synthesis
7. Summing a list of integer values
8. Stategy 3 : The Recursive Interpretation of call
9. Two examples
   - Computing the square of a number
   - Searching a key in an array

Global view of the development

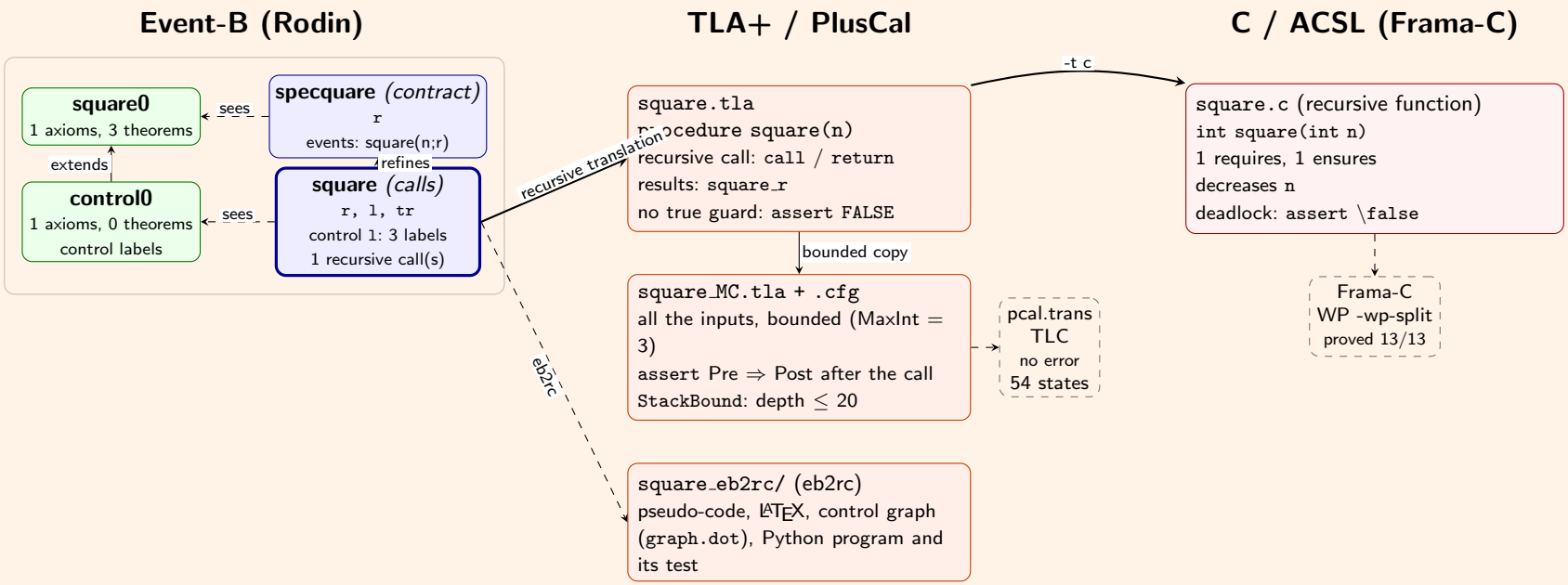
Event-B (Rodin)



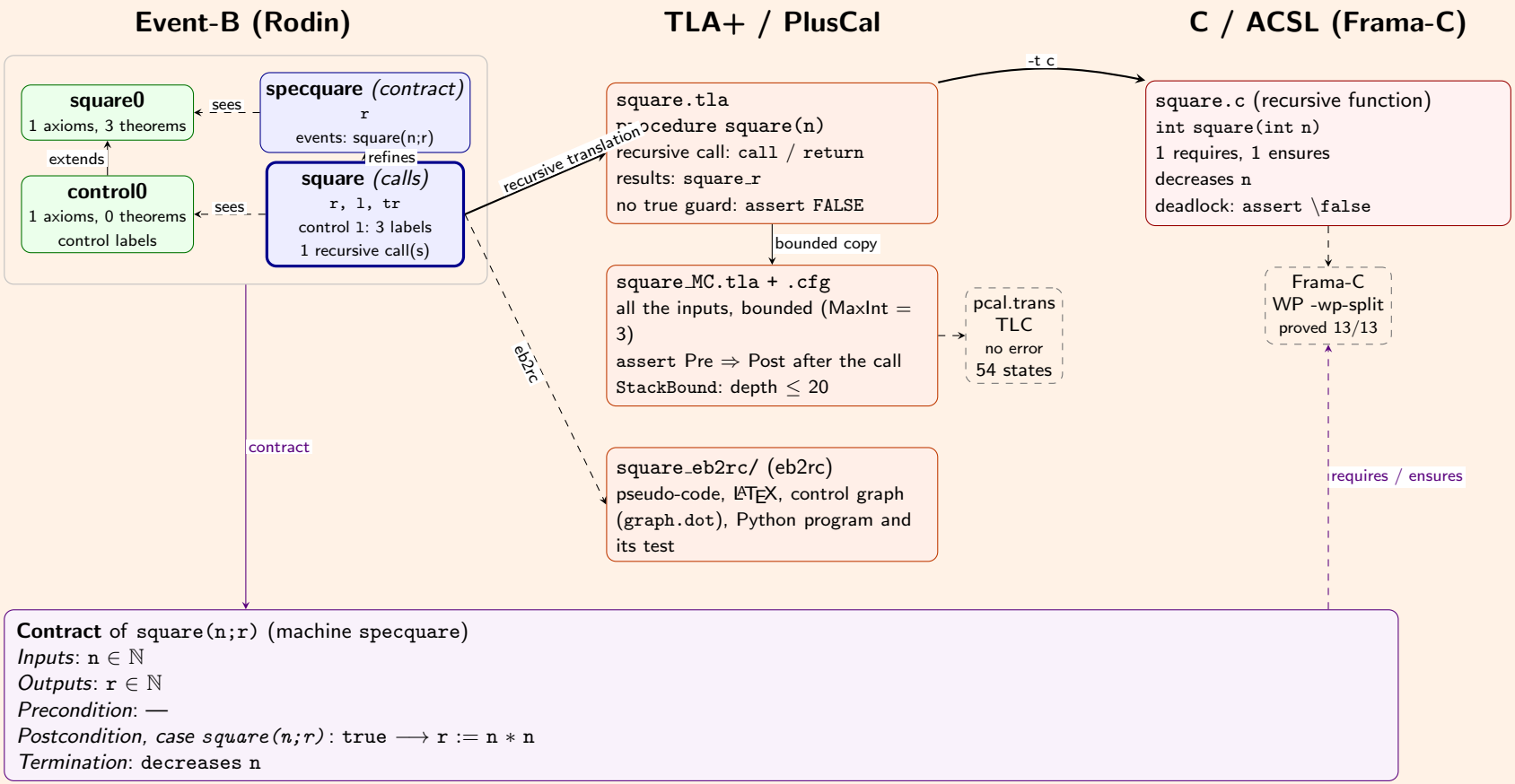
Global view of the development



Global view of the development



Global view of the development



Code for the function

```
/* Recursive function square(n;r) translated by eb2pcal (case rec) from the
   machine square of the project zb-square; contract: machine specquare.
   Check with: frama-c -wp -wp-split -wp-prover alt-ergo,z3 square.c */

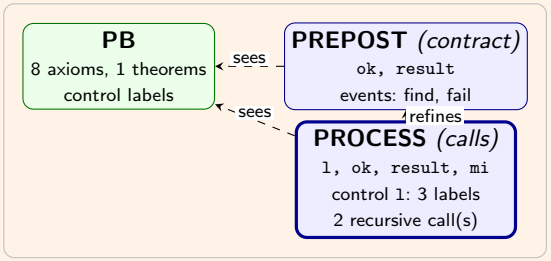
/*@
  requires (0 <= n);    // typage de n
  decreases n;
  assigns \nothing;
  ensures ((\result == (n * n)));
  // square(n;r)
*/
int square(int n)
{
    int tr = 0;
    int r = 0;
    if ((n == 0)) {
        r = 0;
    } else if ((n != 0)) {
        tr = square((n - 1));
        r = ((tr + (2 * (n - 1))) + 1);
    } else {
        /*@ assert \false; */  /* no guard true: deadlock */
    }
    return r;
}
```

Current Sub Section Summary

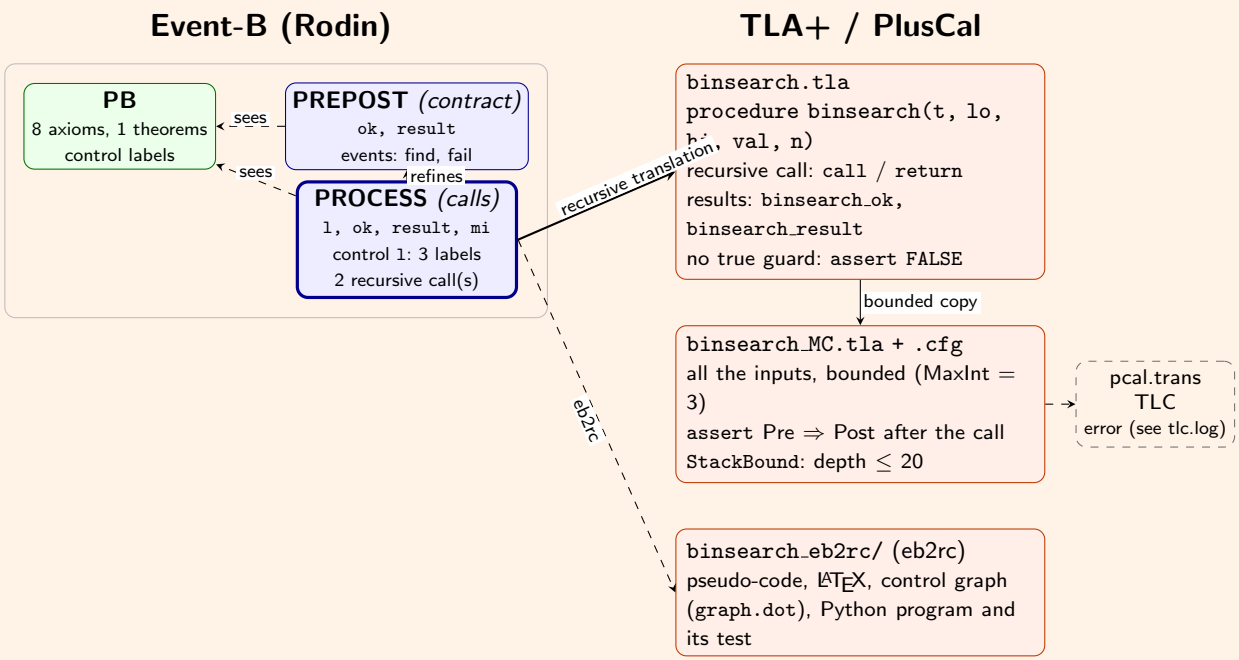
1. CM5
2. Overview
3. Programming by contract
4. Strategy 1 : Transformations of the ALGO machine
5. Strategy 2 : Internalizing events by merging
6. Analysis and then Synthesis
7. Summing a list of integer values
8. Stategy 3 : The Recursive Interpretation of call
9. Two examples
   - Computing the square of a number
   - Searching a key in an array

Global view of the development

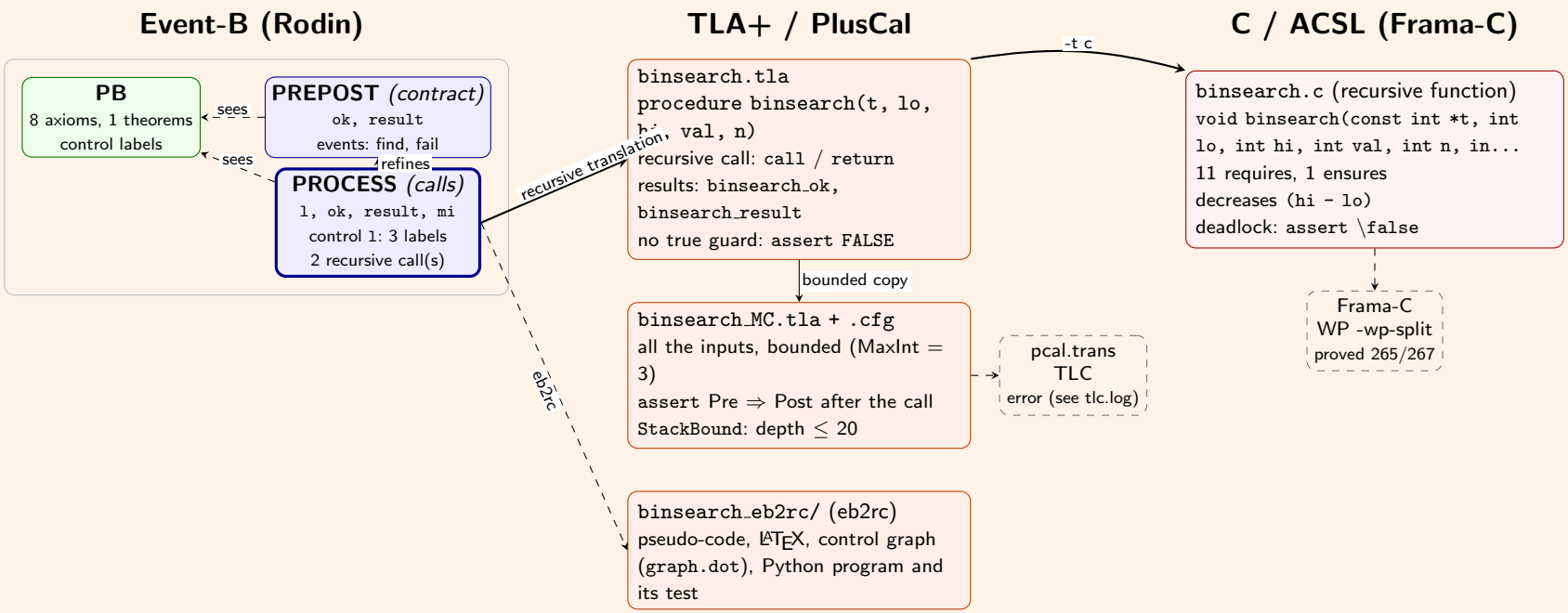
Event-B (Rodin)



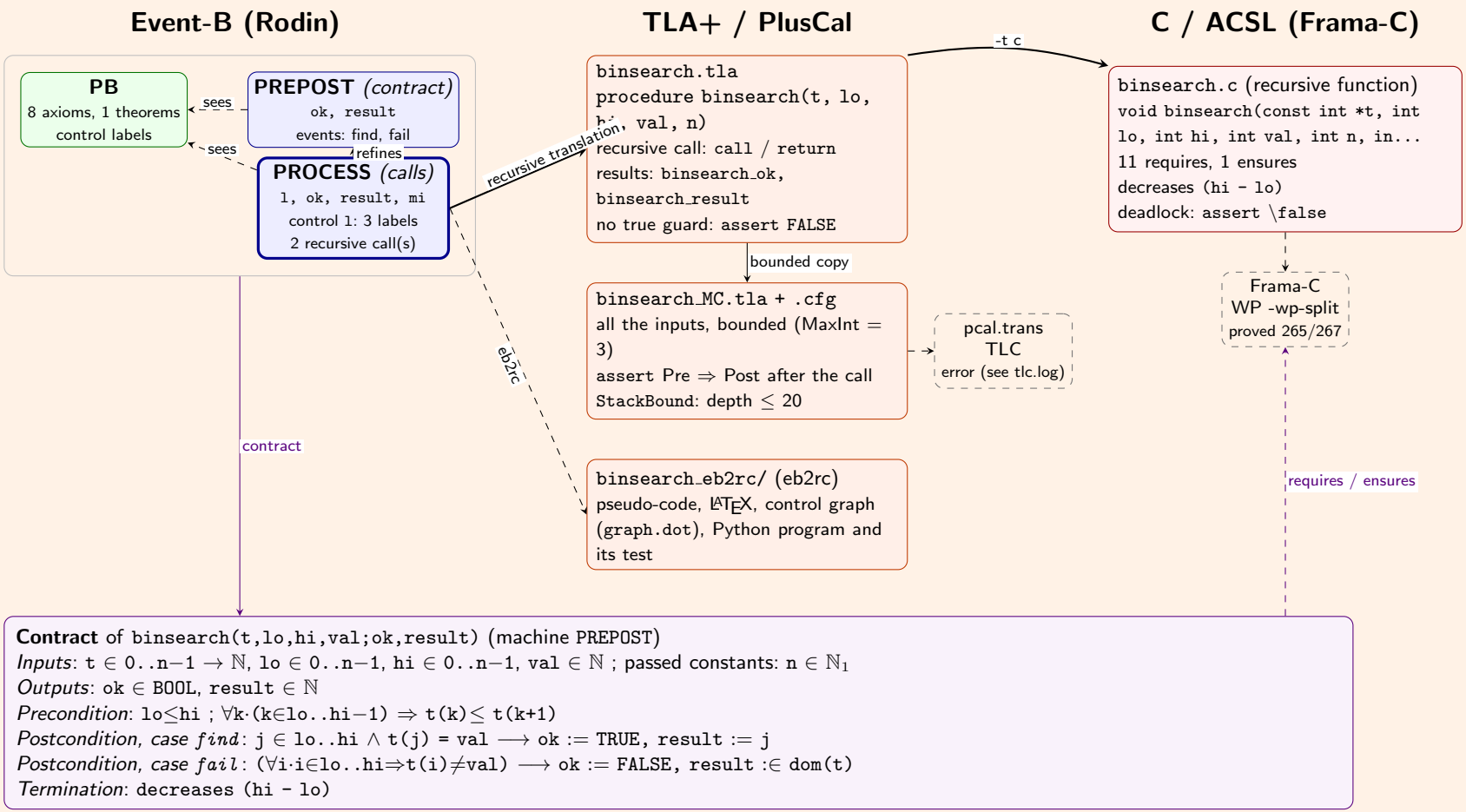
Global view of the development



Global view of the development



Global view of the development



Code for the function

```
/* Recursive function binsearch(t,lo,hi,val;ok,result) translated by eb2pca
   machine PROCESS of the project zb-binarysearch; contract: machine PREPOS
   Check with: frama-c -wp -wp-split -wp-prover alt-ergo,z3 binsearch.c */

/*@ lemma PB_axm7_global{L}:   // RR13 : Ã  prouver par rÃ©currence
      \forall int *t, integer a, b;
        (\forall integer k; a <= k <= b - 1 ==> t[k] <= t[k + 1]) ==>
        \forall integer i, j; a <= i <= j <= b ==> t[i] <= t[j];
*/

/*@
  requires \valid_read(t + (0 .. (n - 1)));   //
  requires \forall integer k_; 0 <= k_ <= (n - 1) ==> (0 <= t[k_]);
//
  requires (0 <= lo && lo <= (n - 1));   // typage de lo
  requires (0 <= hi && hi <= (n - 1));   // typage de hi
  requires (0 <= val);   // typage de val
  requires (1 <= n);   // typage de n
  requires (lo <= hi);   // PB.axm6
  requires (\forall integer k; ((lo <= k && k <= (hi - 1)) ==> (t[k] <= t[(
// PB.axm7
  requires \forall integer i_, j_; lo <= i_ <= j_ <= hi ==> t[i_] <= t[j_];
// PB.axm7 (forme globale, RR13)
  requires \valid(ok) && \valid(result);   // sorties
  requires \separated(ok, result, t + (0 .. (n - 1)));   // sorties sÃ©parÃ
  decreases (hi - lo);
  assigns *ok, *result;
  ensures (\exists integer j; (lo <= j && j <= hi) && (t[j] == val) && (*ok
       || ((\forall integer i; ((lo <= i && i <= hi) ==> (t[i] != val))) &&
// find / fail
```