Modelling Recursion in Event-B

Stefan Hallerstede

Universität Düsseldorf

Rodin User and Developer Workshop
Düsseldorf
21 September 2010

Contents

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Contents

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Development and Verification of Recursive Programs

- We seek a method:
  - Easy to apply (also to larger programs)
  - Tool support (such as Rodin)
  - Data refinement
  - Imperative programs and procedures
- Some related work:[1]
  - Program Verification (Hoare)
  - Proof Outlines (Owicki/Gries)
  - Refinement Calculus (Morgan)
  - Refinement Calculus (Back)

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

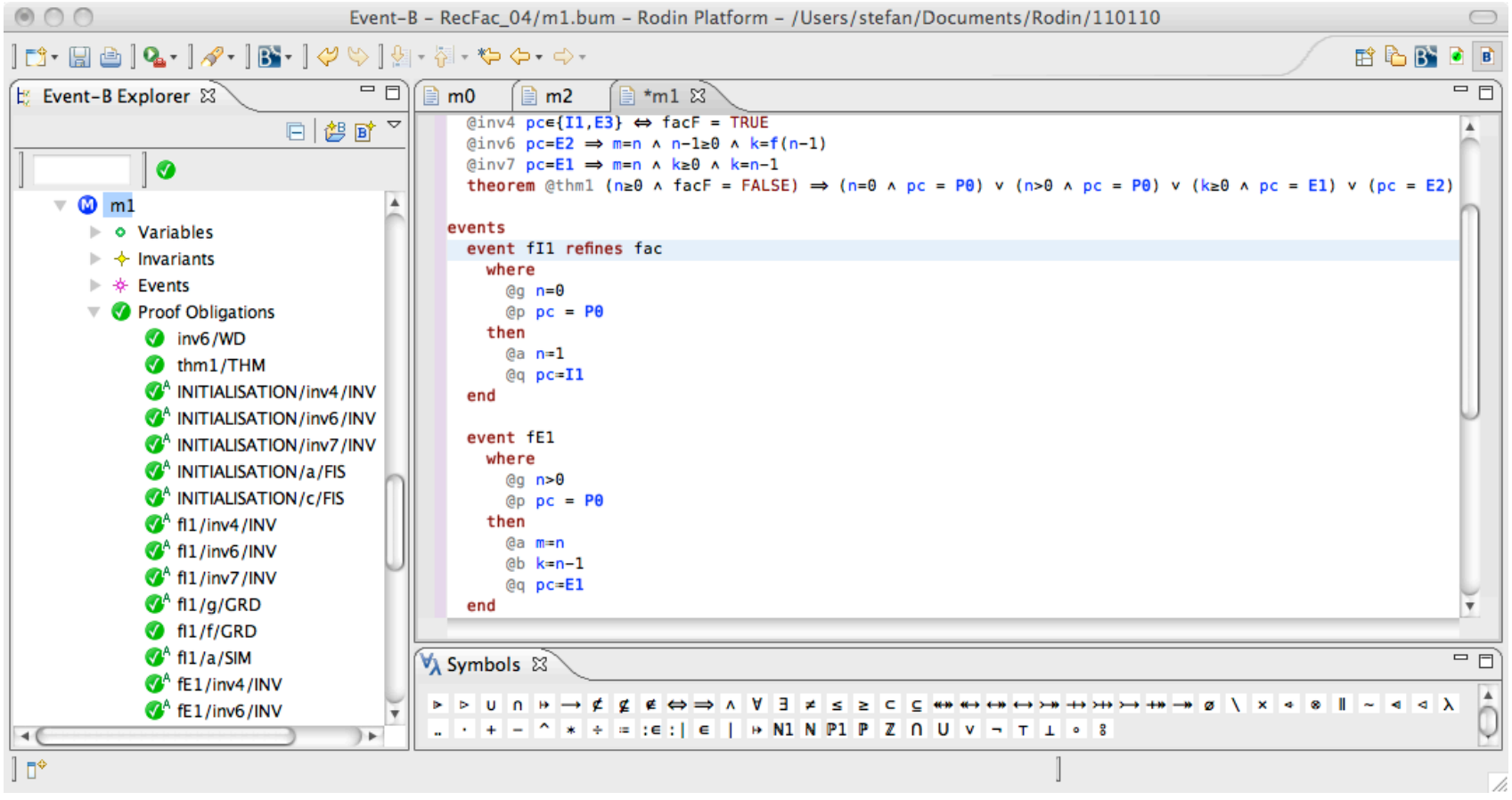
[1] Missing from the list: B-Method and VDM, for instance

Method of Program Development

- A model describes an "*abstract program*"
- "*Concrete programs*" are an indirect target
  - Maybe not reached by refinement
- "*Abstract program*":
  - only call-by-reference parameters
  - and no global variables

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Use of Event-B (and Rodin)



- Clear and direct mapping to Event-B machines
- Use **Rodin** to do the proofs
- Keep **Event-B** notion of refinement
- For now write the "mapped" model directly in Event-B

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Contents

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Proof Outline of Factorial Procedure

Program Verification

$\{\, n \geq 0 \wedge n = N \,\}$ *Fac*(n) $\{\, n = N! \,\}$ ::

$\{\, n \geq 0 \wedge n = N \,\}$

if $n = 0$ then

$\{\, n = 0 \wedge n = N \,\}$

$n := 1$

$\{\, n = N! \,\}$

else

$\{\, n > 0 \wedge n = N \,\}$

var $m := n$;

$\{\, m = N \wedge n > 0 \wedge n = N{-}1 \,\}$

$n := n{-}1$;

$\{\, m = N \wedge n \geq 0 \wedge n = (N{-}1) \,\}$

Fac(n);

$\{\, m = N \wedge n = (N{-}1)! \,\}$

$n := m * n$

$\{\, n = N! \,\}$

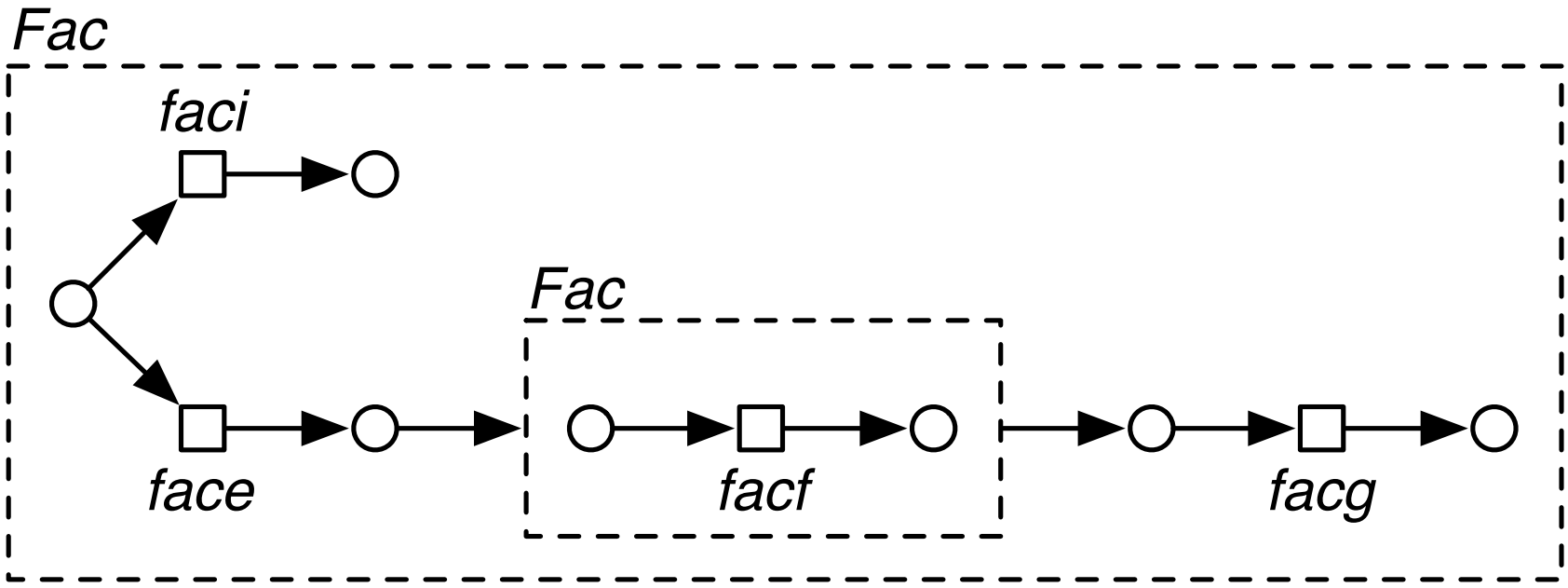
end

$\{\, n = N! \,\}$

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Graphical Notation for a Proof Outline Using Events

Towards Event-B



- *faci* $=$ when $n = 0$ then $n := 1$ end
- *face* $=$ when $n > 0$ then $m, n := n, n - 1$ end
- *facf* $=$ when $n \geq 0$ then $n := n!$ end
- *facg* $=$ $n := m * n$

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

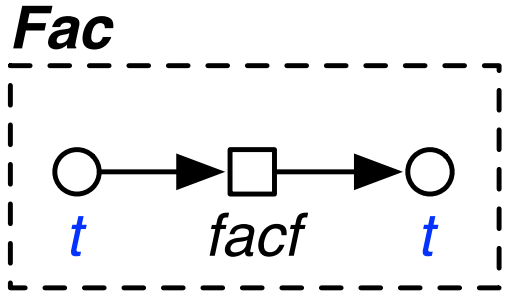
Contents

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Contents

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

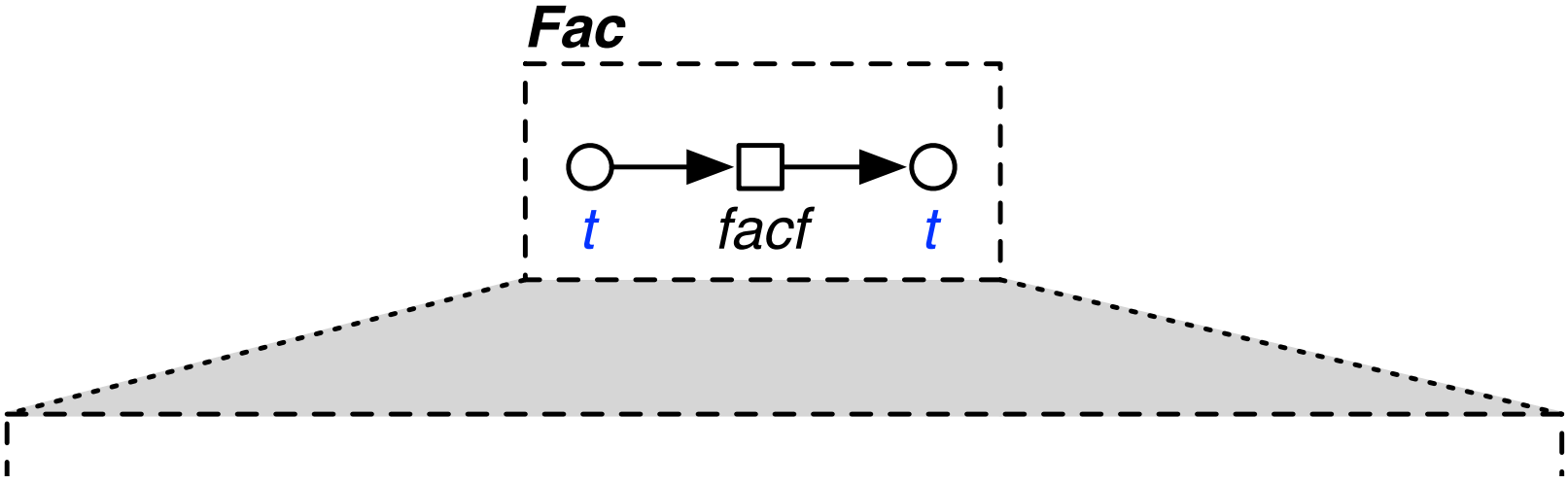
Factorial Specification



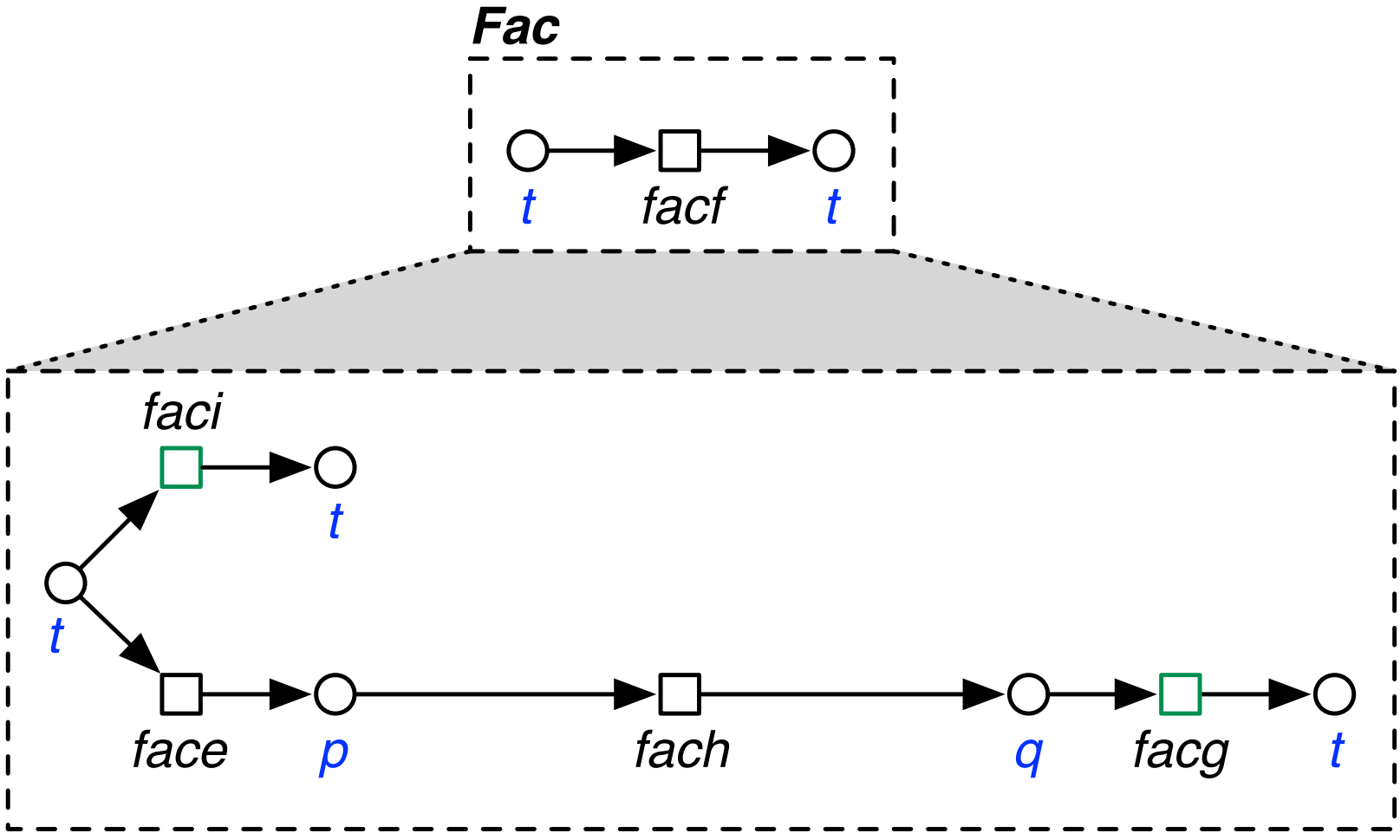
@t *true*

facf $=$ when $n \geq 0$ then $n := n!$ end

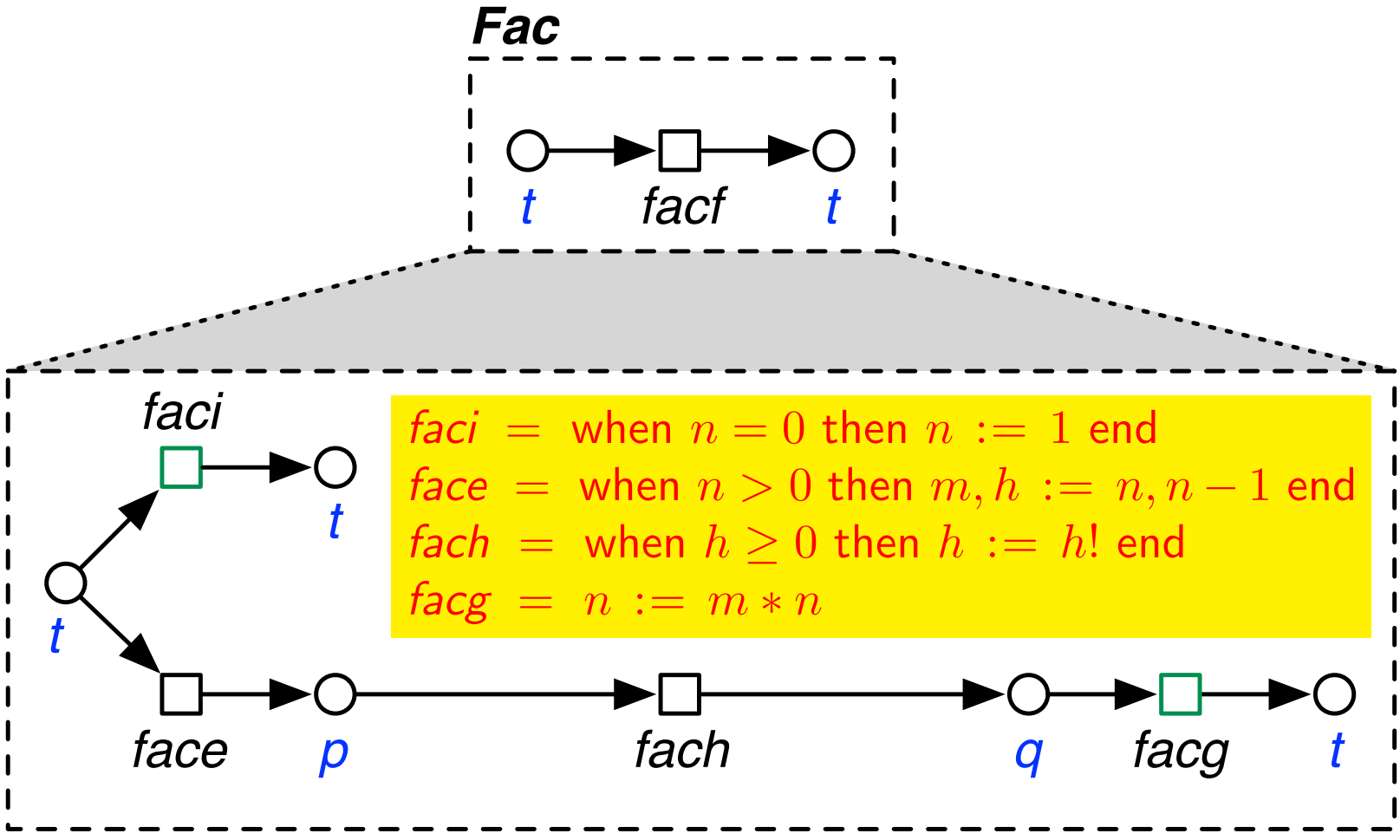
Factorial Refinement Sketch



Factorial Refinement Sketch

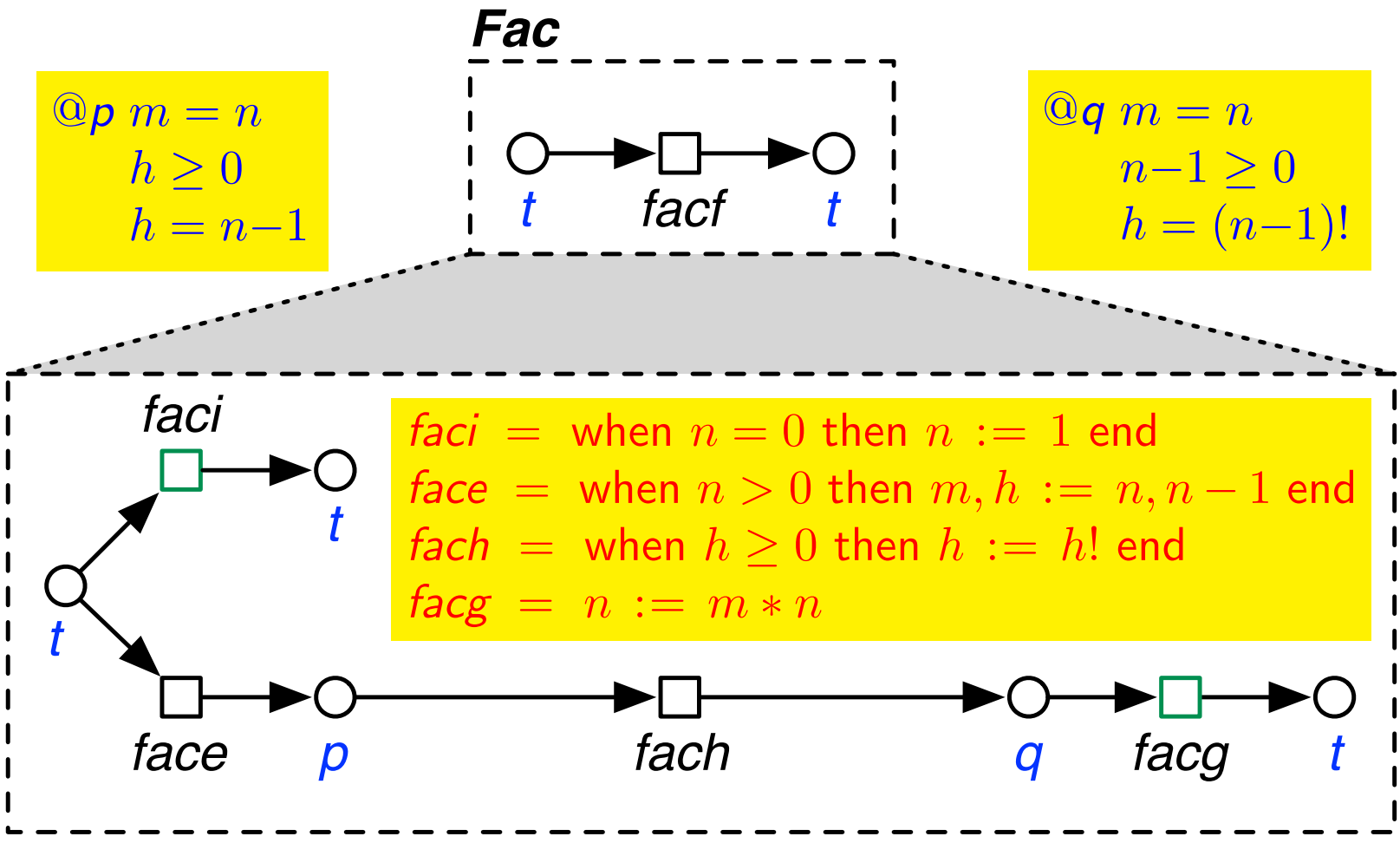


Factorial Refinement Sketch



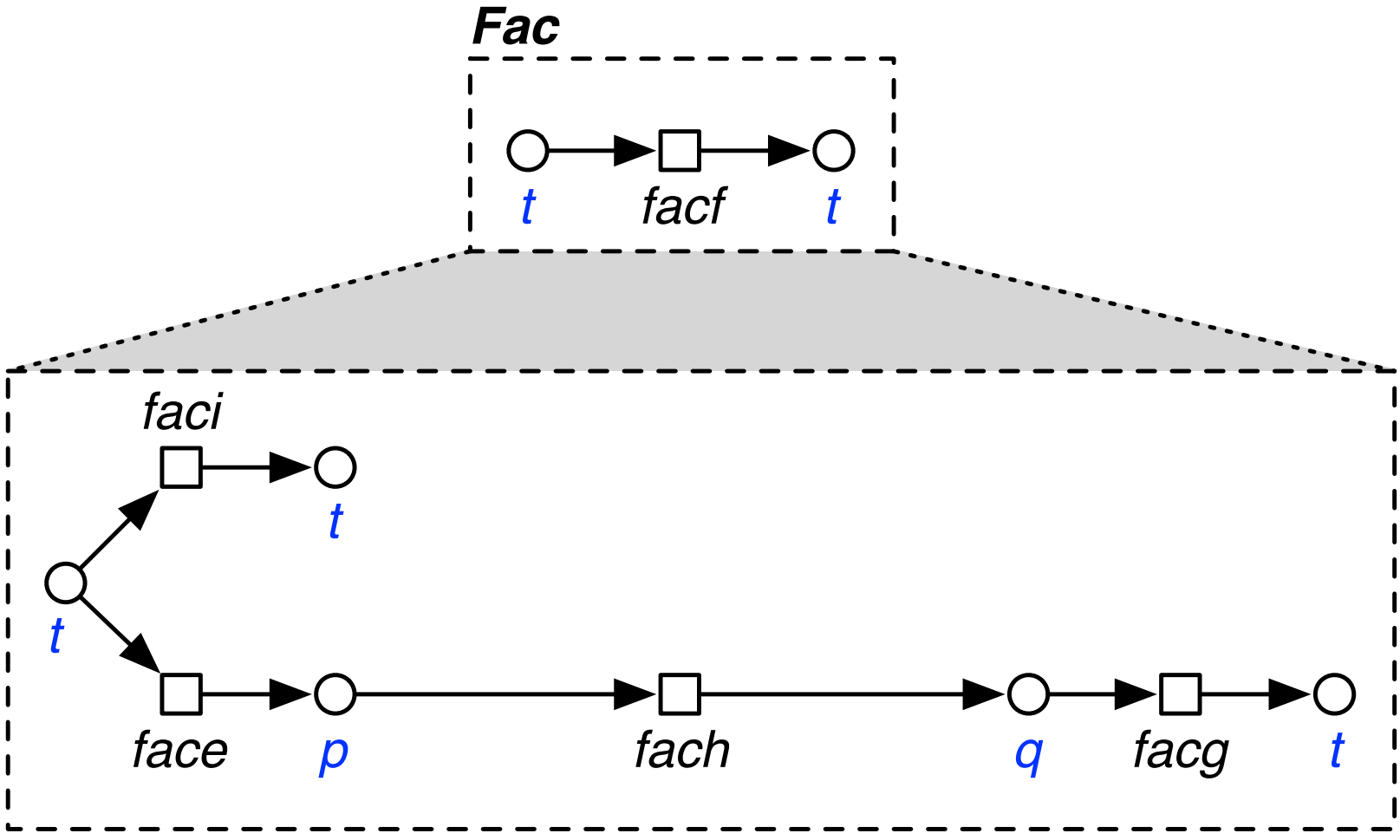
HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Factorial Refinement Sketch



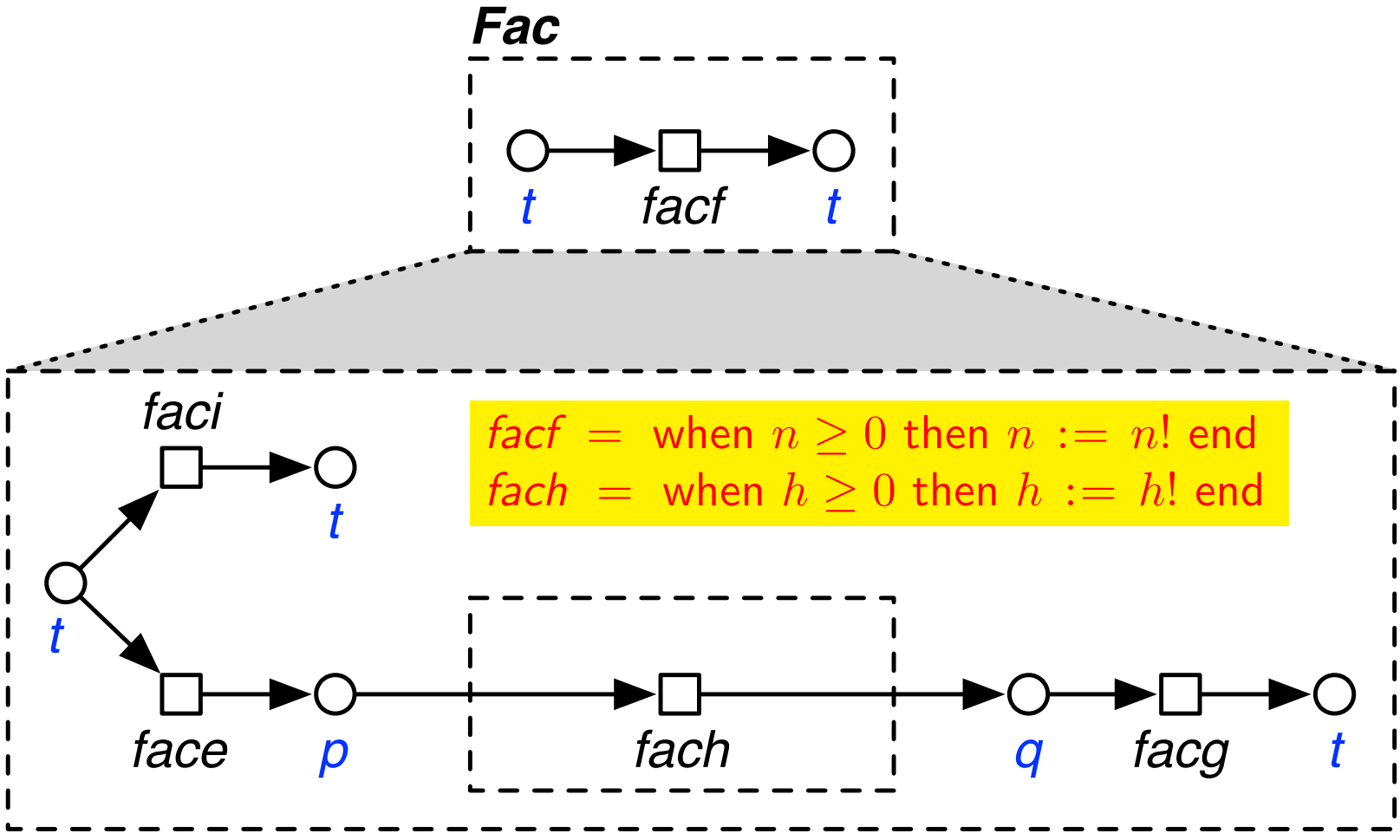
HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Factorial Refinement Sketch



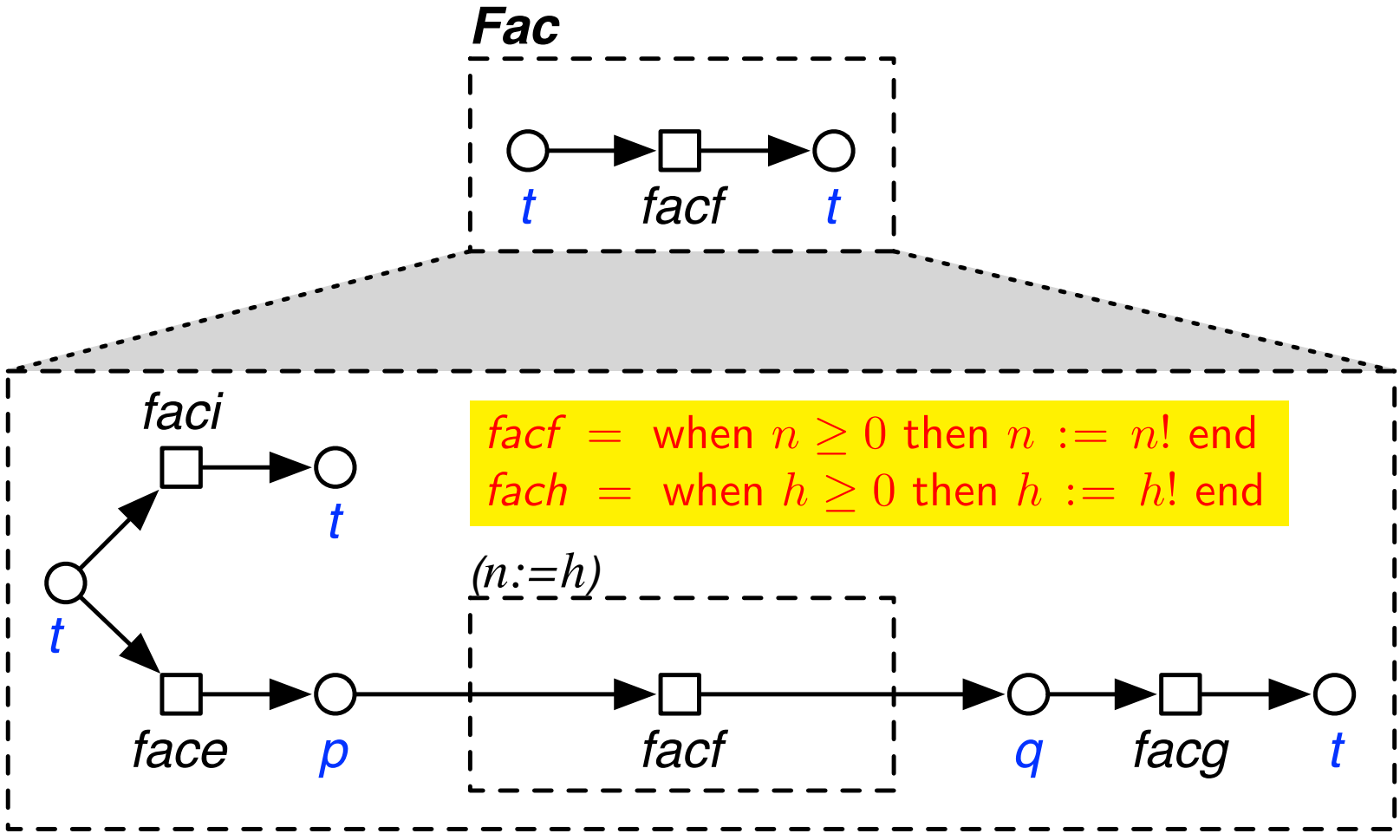
HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

A Recursive Reference?



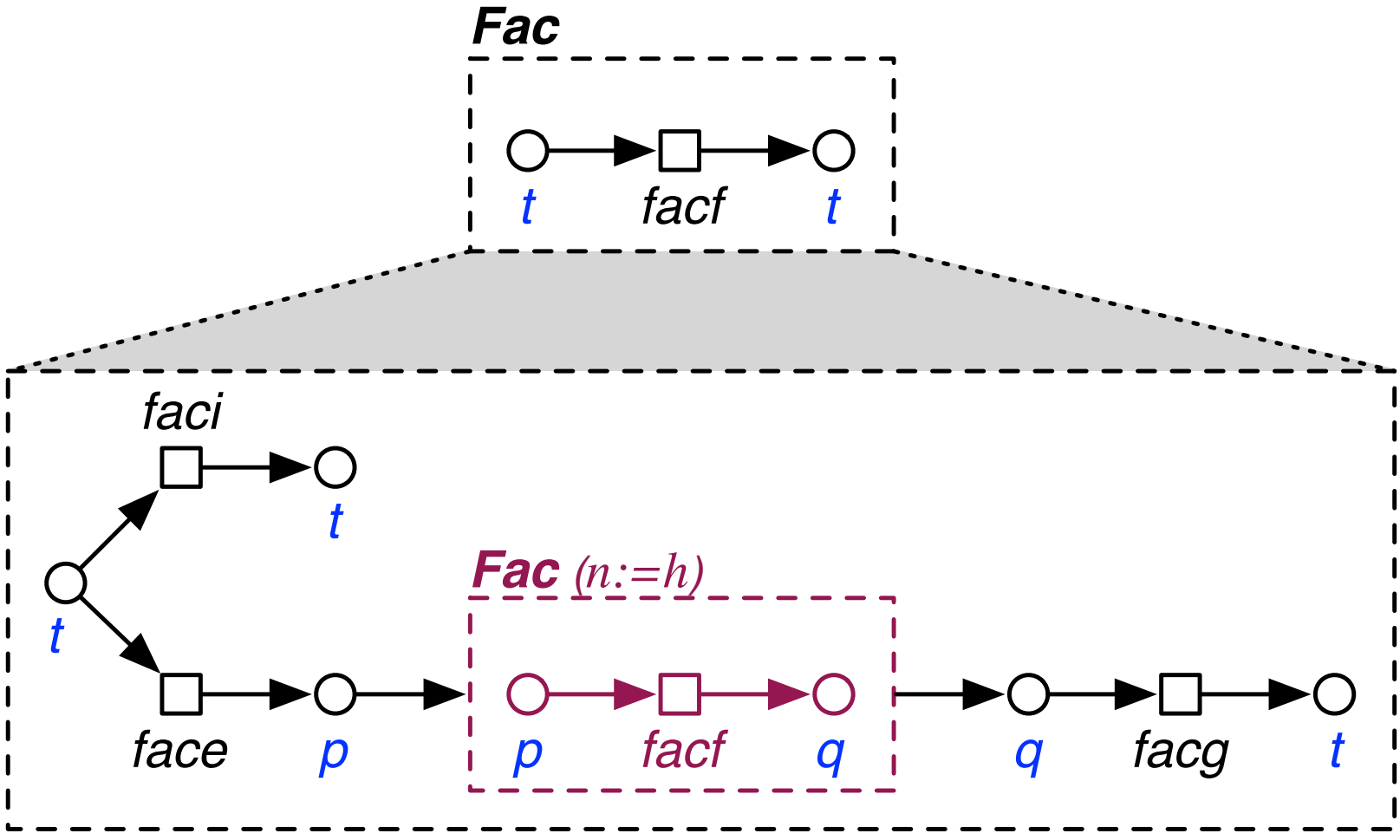
HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

A Recursive Reference?

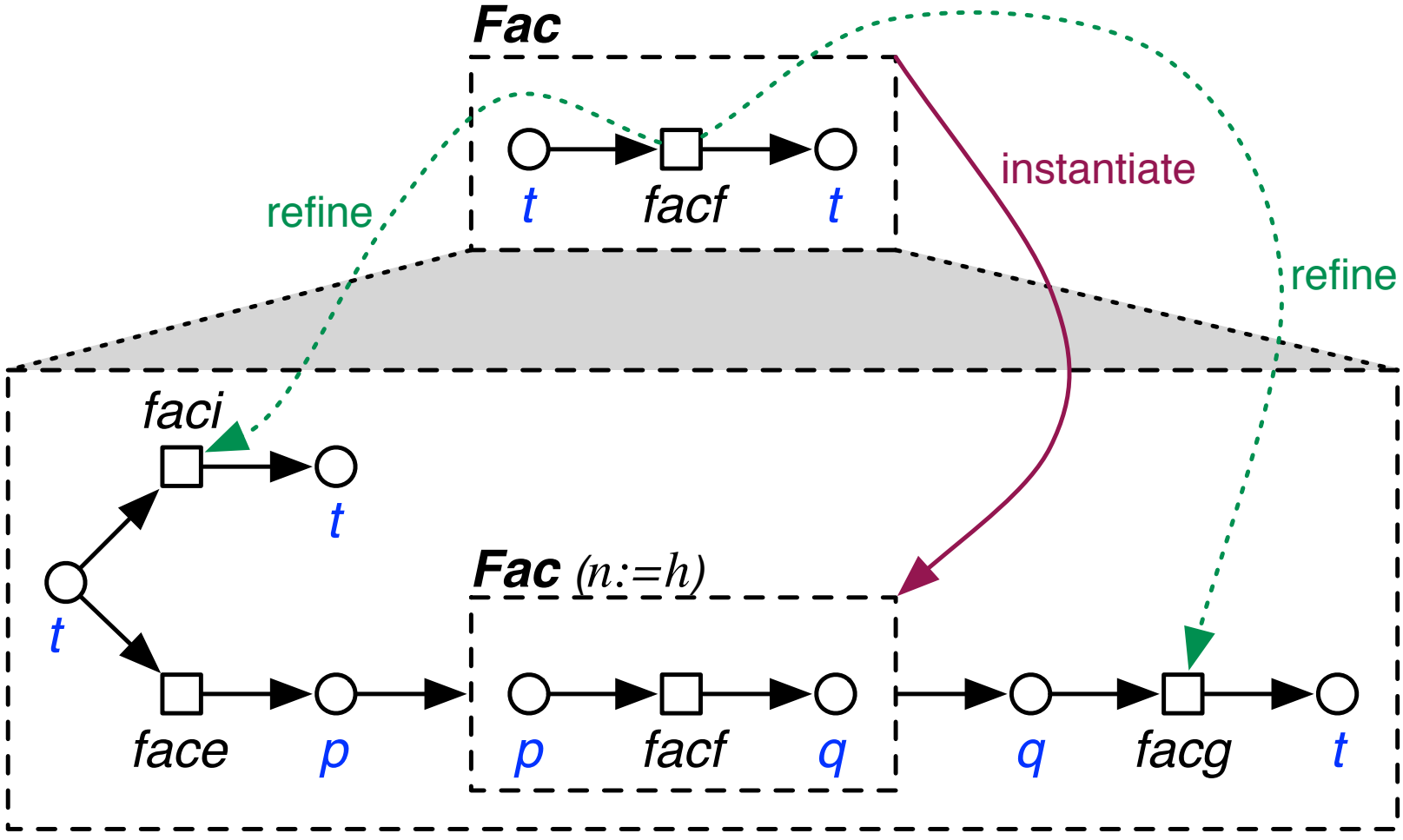


HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Recursion by Instantiation



Completed Factorial Model

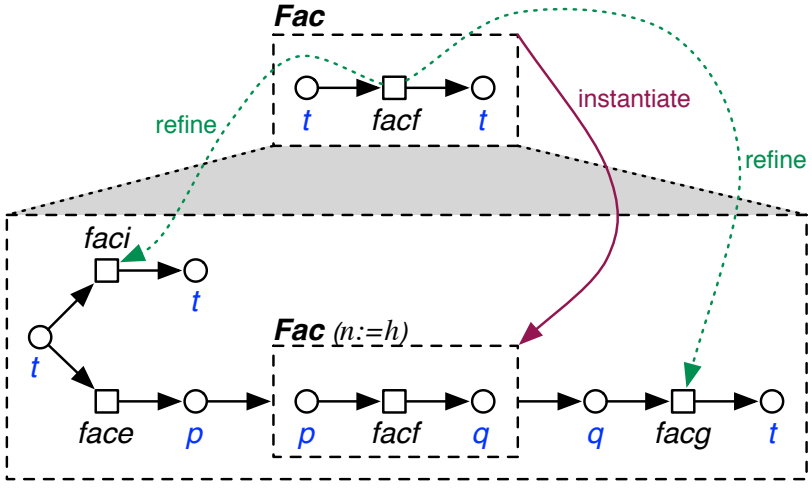


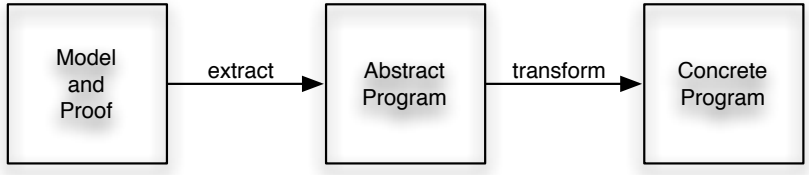
HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Contents

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

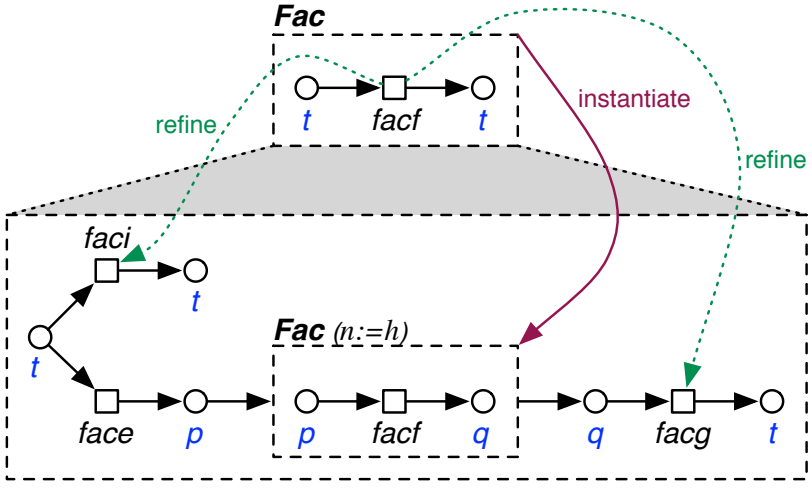
Extracted Factorial Program

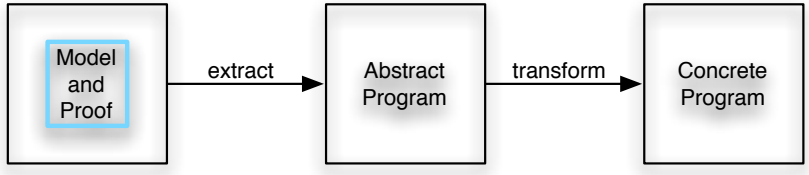




HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

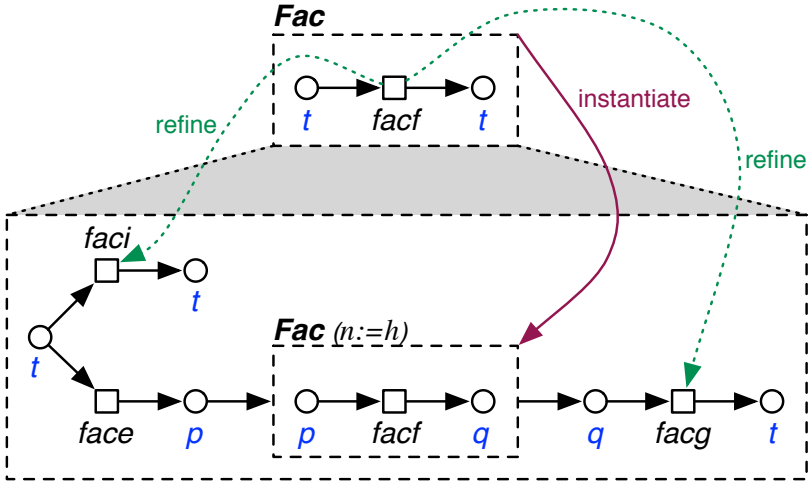
Extracted Factorial Program

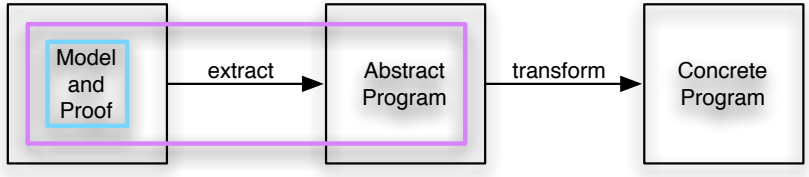




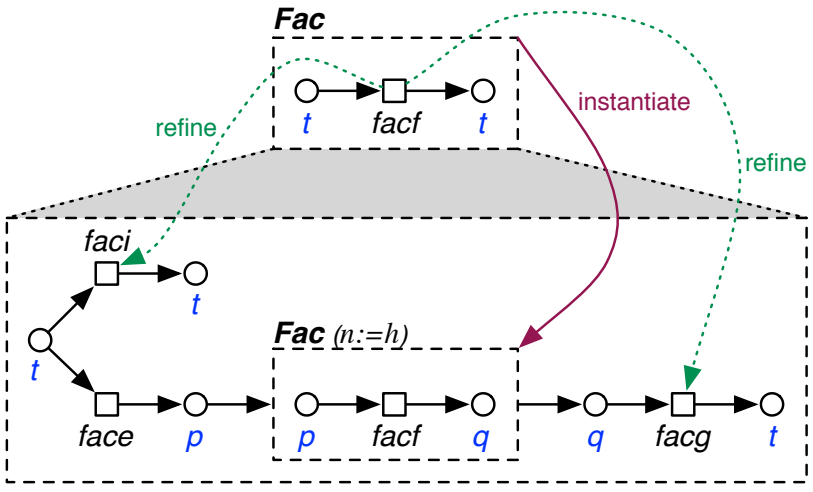
HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Extracted Factorial Program

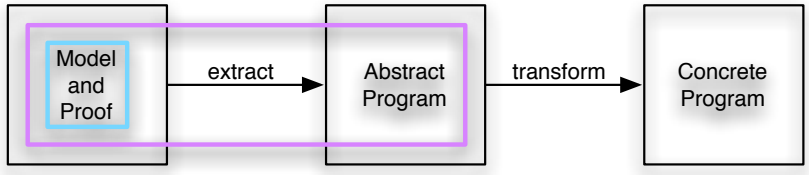




Extracted Factorial Program

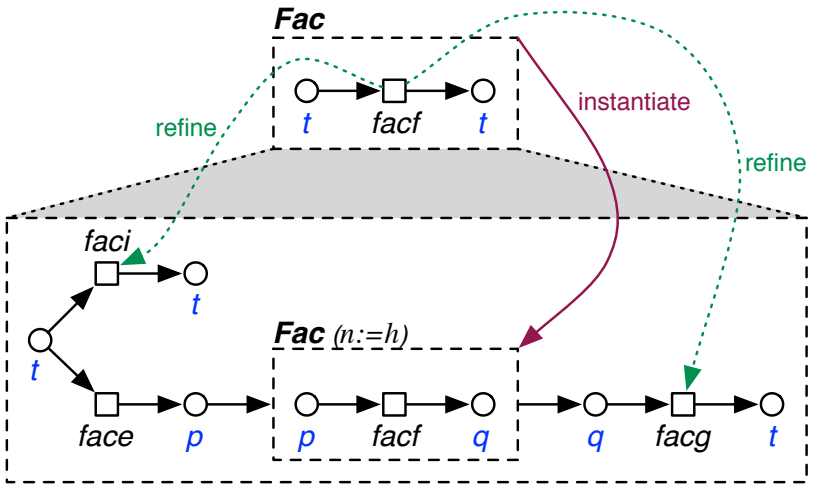


$$
\begin{array}{l}
\mathit{Fac}(n) \ :: \\
\quad \text{if } n = 0 \text{ then} \\
\qquad n \ := \ 1 \\
\quad \text{else} \\
\qquad \text{var } m,\, h \ := \ n,\, n{-}1\,; \\
\qquad \mathit{Fac}(h)\,; \\
\qquad n \ := \ m * h \\
\quad \text{end}
\end{array}
$$

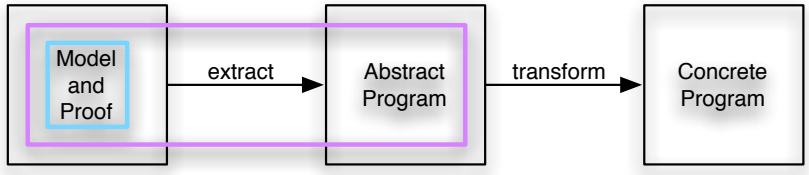


HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Extracted Factorial Program

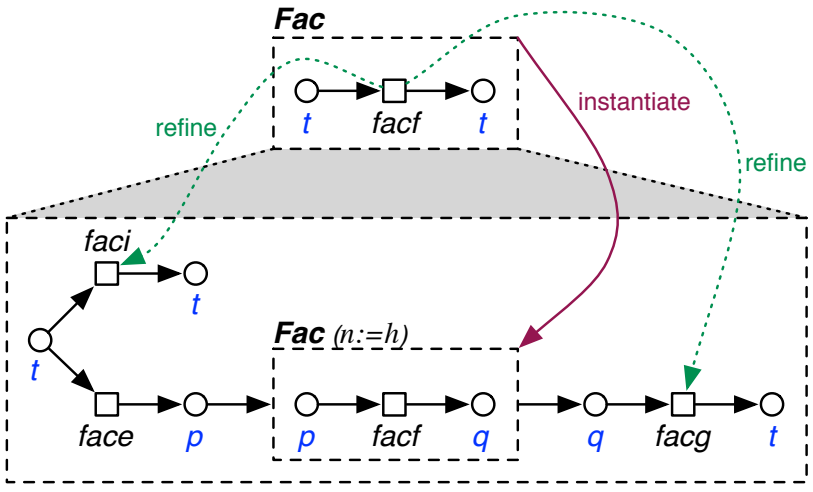


$$
\begin{array}{l}
\mathit{Fac}(n) \ :: \\
\quad \text{if } n = 0 \text{ then} \\
\qquad n \ := \ 1 \\
\quad \text{else} \\
\qquad \text{var } m,\, h \ := \ n,\, n{-}1\,; \\
\qquad \mathit{Fac}(h)\,; \\
\qquad n \ := \ m * h \\
\quad \text{end}
\end{array}
$$

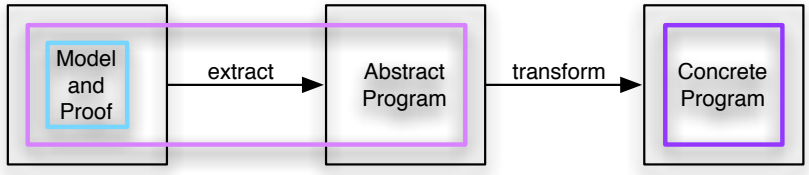


HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Extracted Factorial Program



$$
\begin{array}{l}
\mathit{Fac}(n) \ :: \\
\quad \text{if } n = 0 \text{ then} \\
\qquad n \ := \ 1 \\
\quad \text{else} \\
\qquad \text{var } m,\ h \ := \ n,\ n-1\,; \\
\qquad \mathit{Fac}(h)\,; \\
\qquad n \ := \ m * h \\
\quad \text{end}
\end{array}
$$

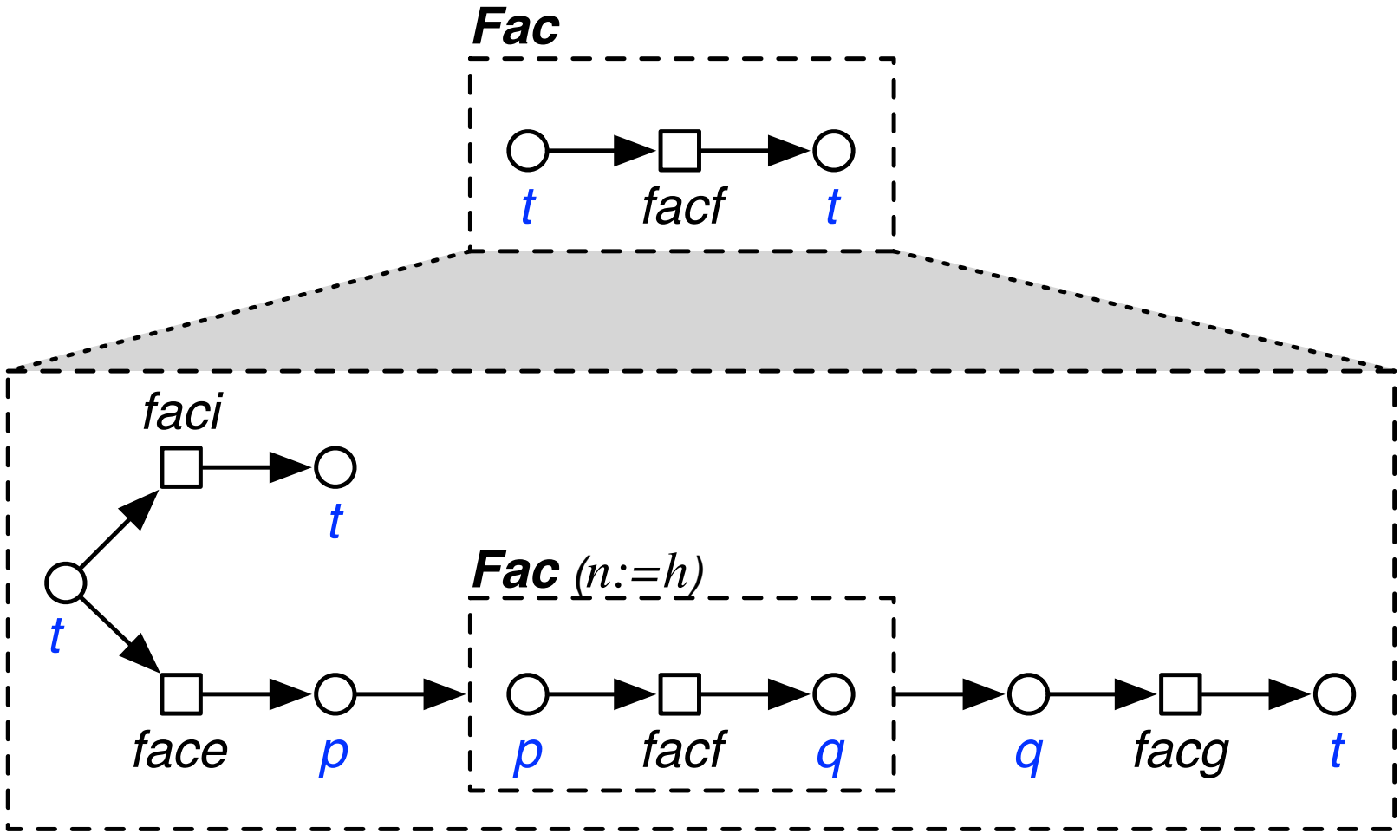


HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Contents

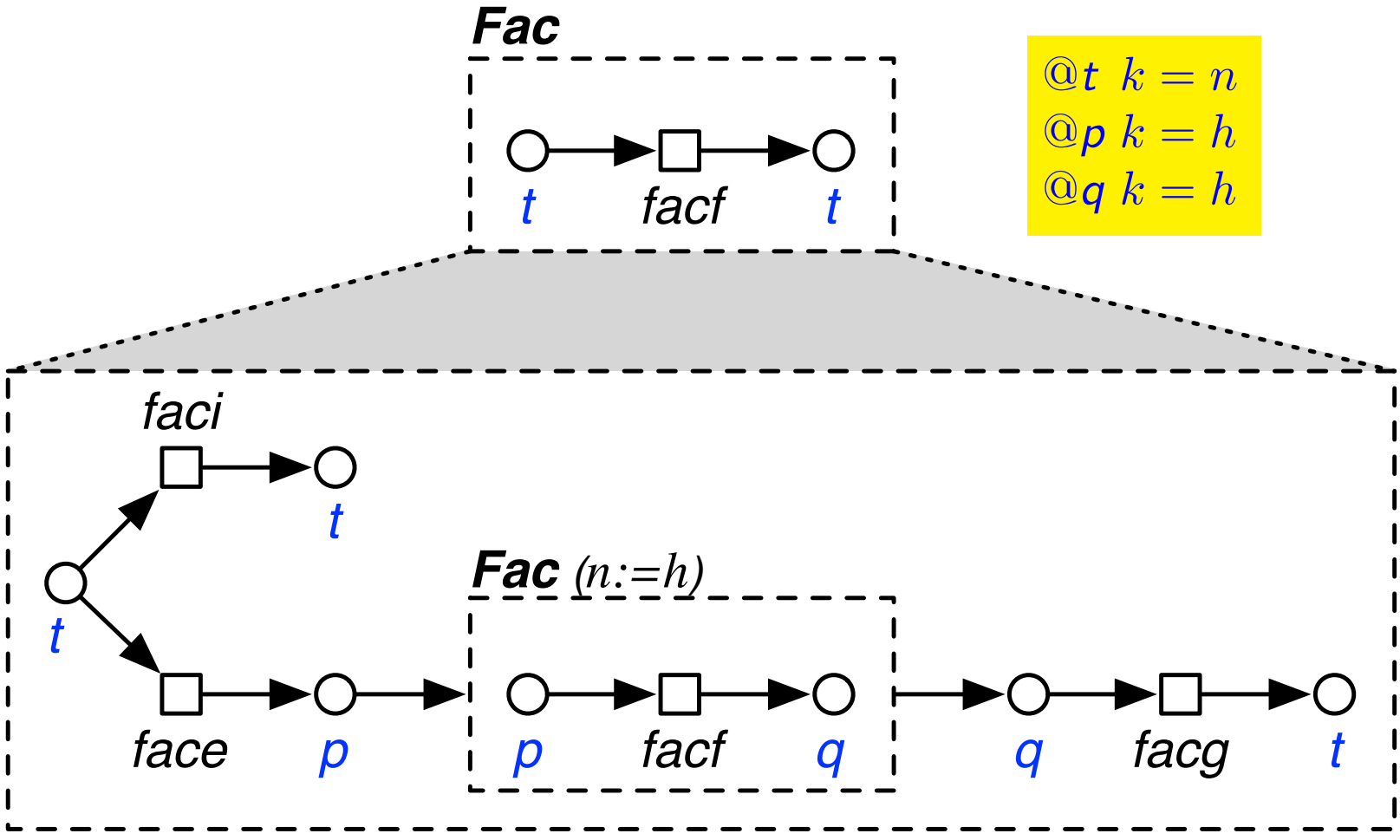
HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Towards a Concrete Program

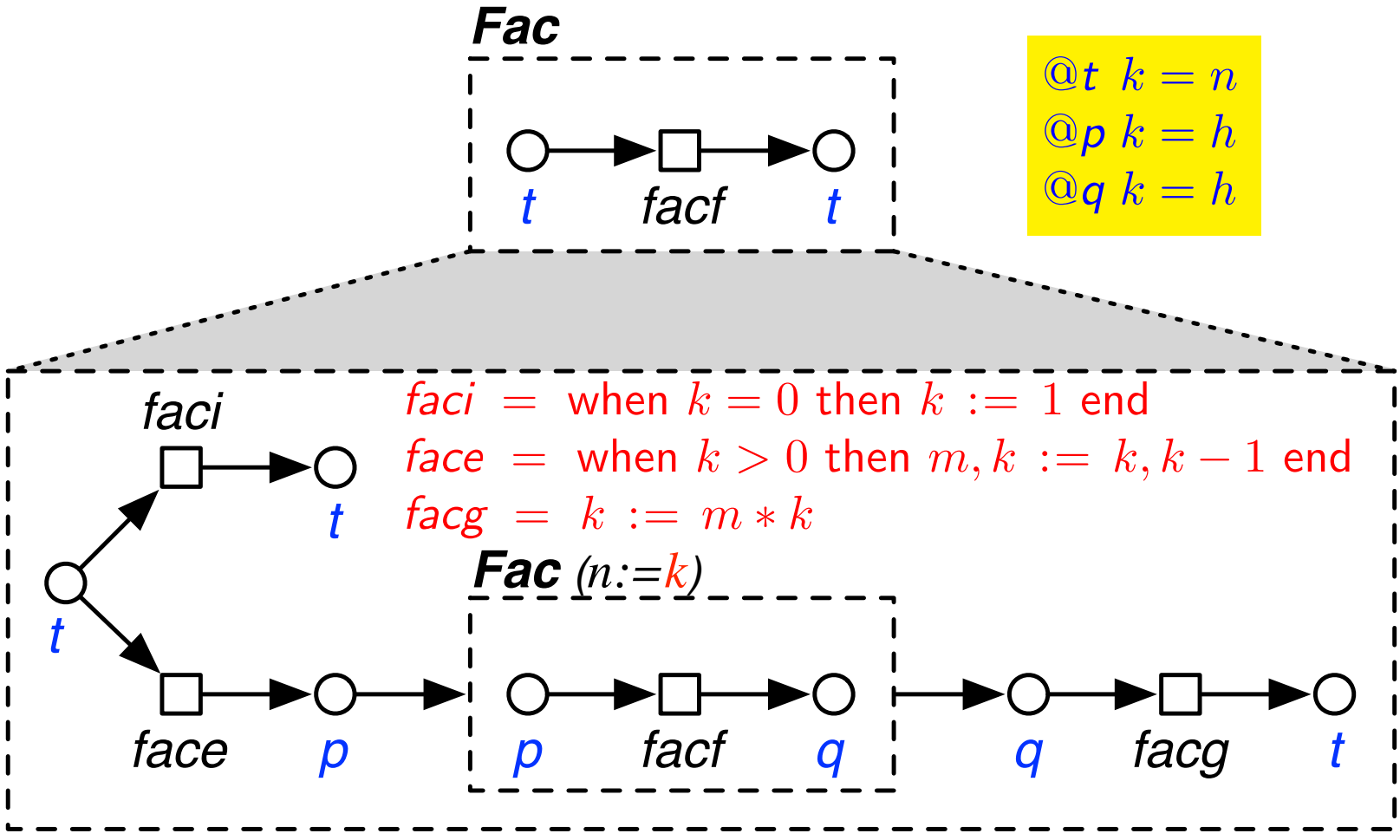


HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Towards a Concrete Program



Towards a Concrete Program



HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Concrete Factorial Program

```
Fac(k) ::
  if k = 0 then
    k := 1
  else
    var m := k;
    k := k−1;
    Fac(k);
    k := m ∗ k
  end
```

Concrete Factorial Program

```
Fac(k) ::
  if k = 0 then
    k := 1
  else
    var m := k;
    k := k−1;
    Fac(k);
    k := m ∗ k
  end
```

transform $\longrightarrow$

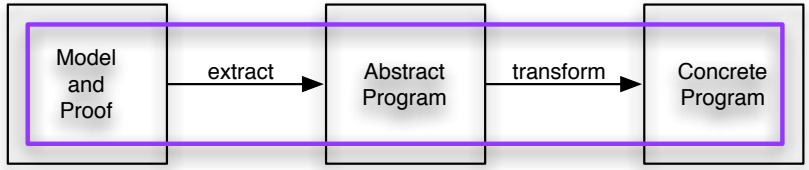
```
global var k
Fac ::
  if k = 0 then
    k := 1
  else
    var m := k;
    k := k−1;
    Fac;
    k := m ∗ k
  end
```

Concrete Factorial Program

```
Fac(k) ::
  if k = 0 then
    k := 1
  else
    var m := k;
    k := k−1;
    Fac(k);
    k := m ∗ k
  end
```

transform $\longrightarrow$

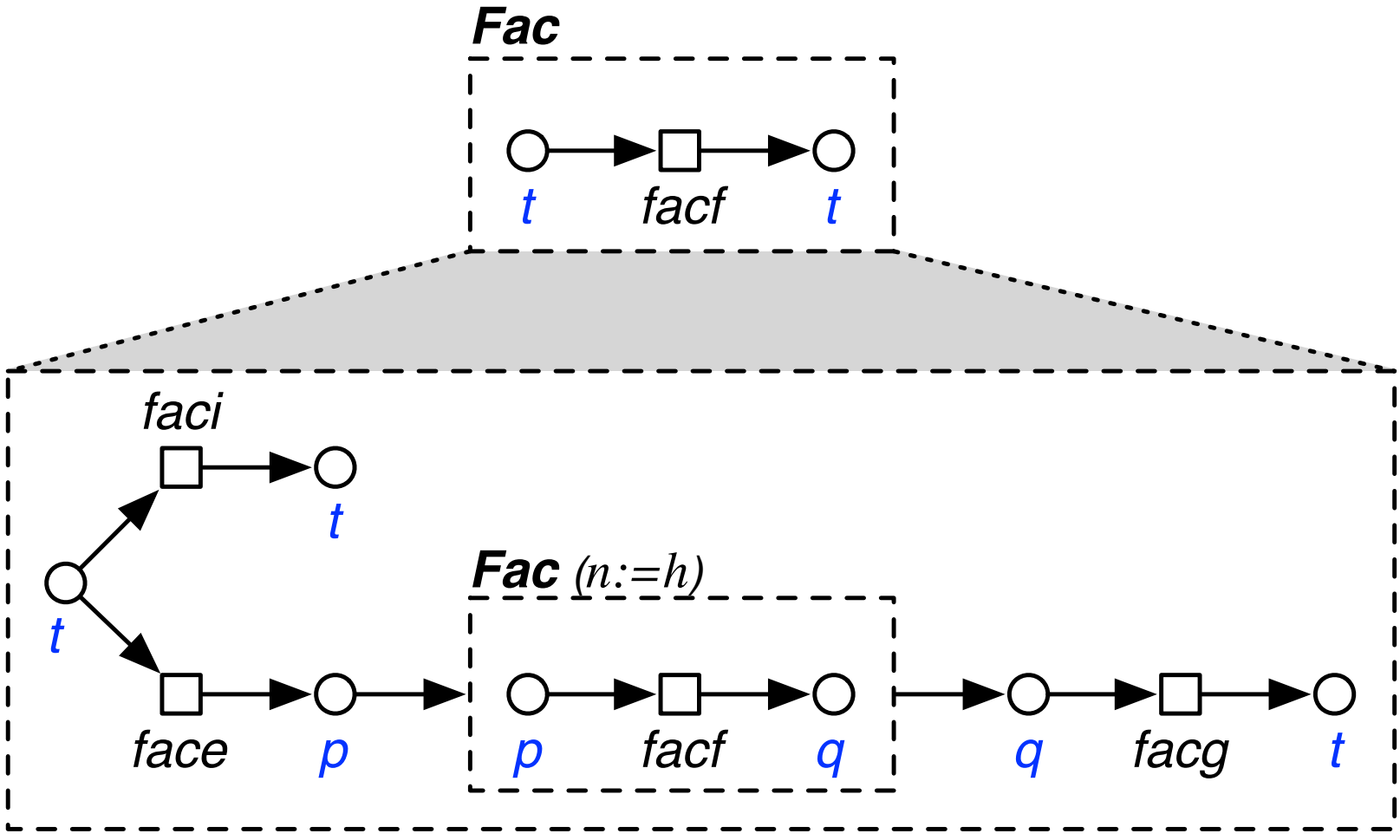
```
global var k
Fac ::
  if k = 0 then
    k := 1
  else
    var m := k;
    k := k−1;
    Fac;
    k := m ∗ k
  end
```



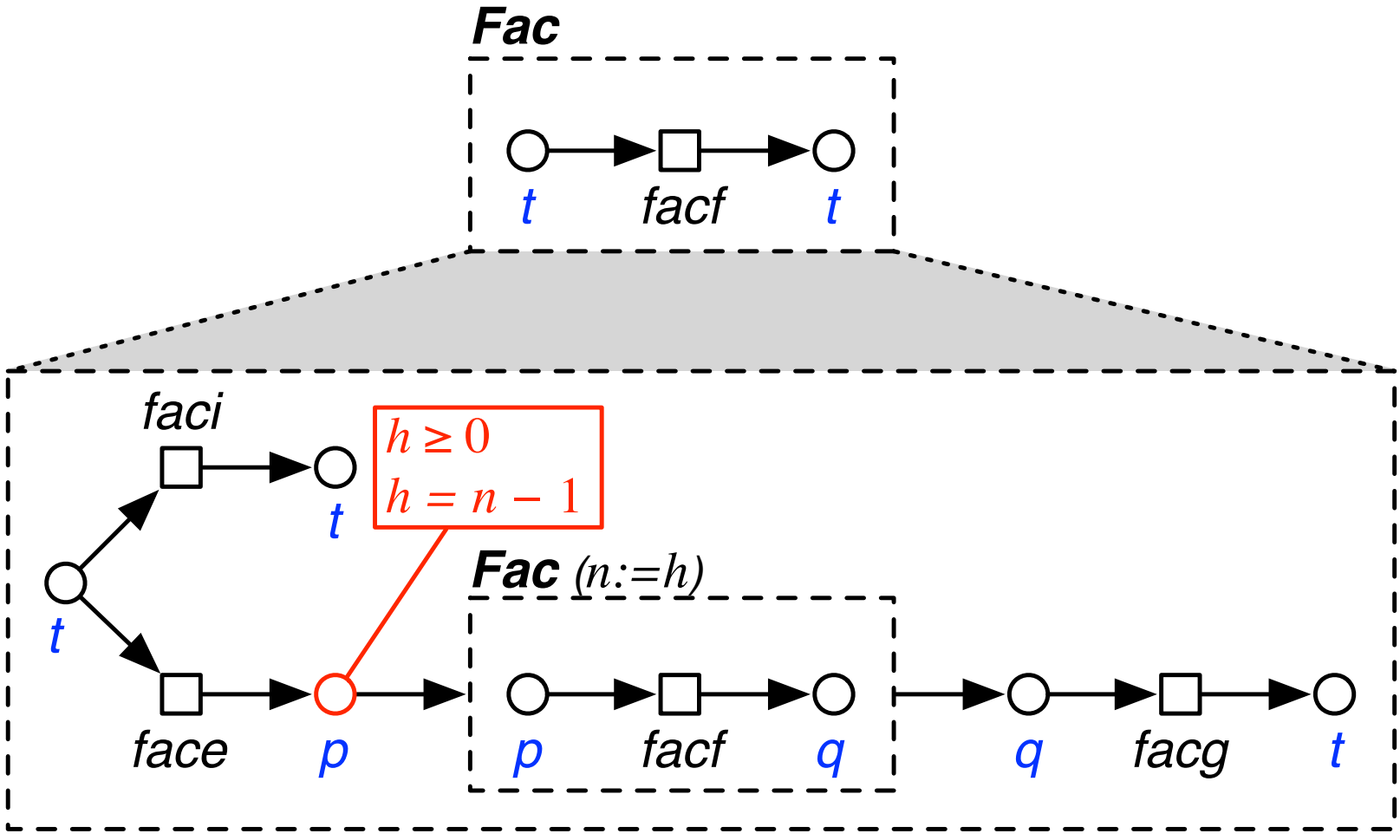
Contents

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

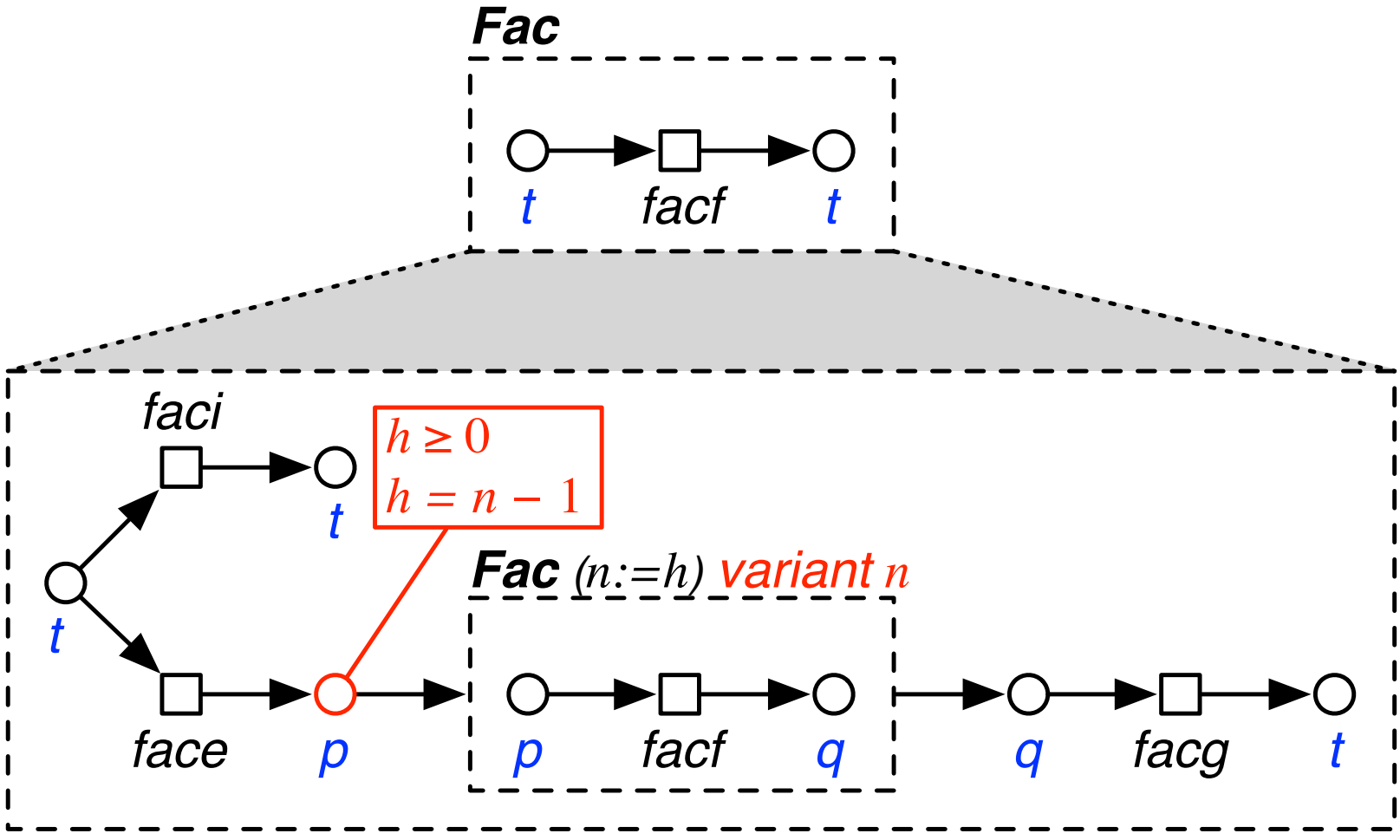
Termination of the Factorial Procedure



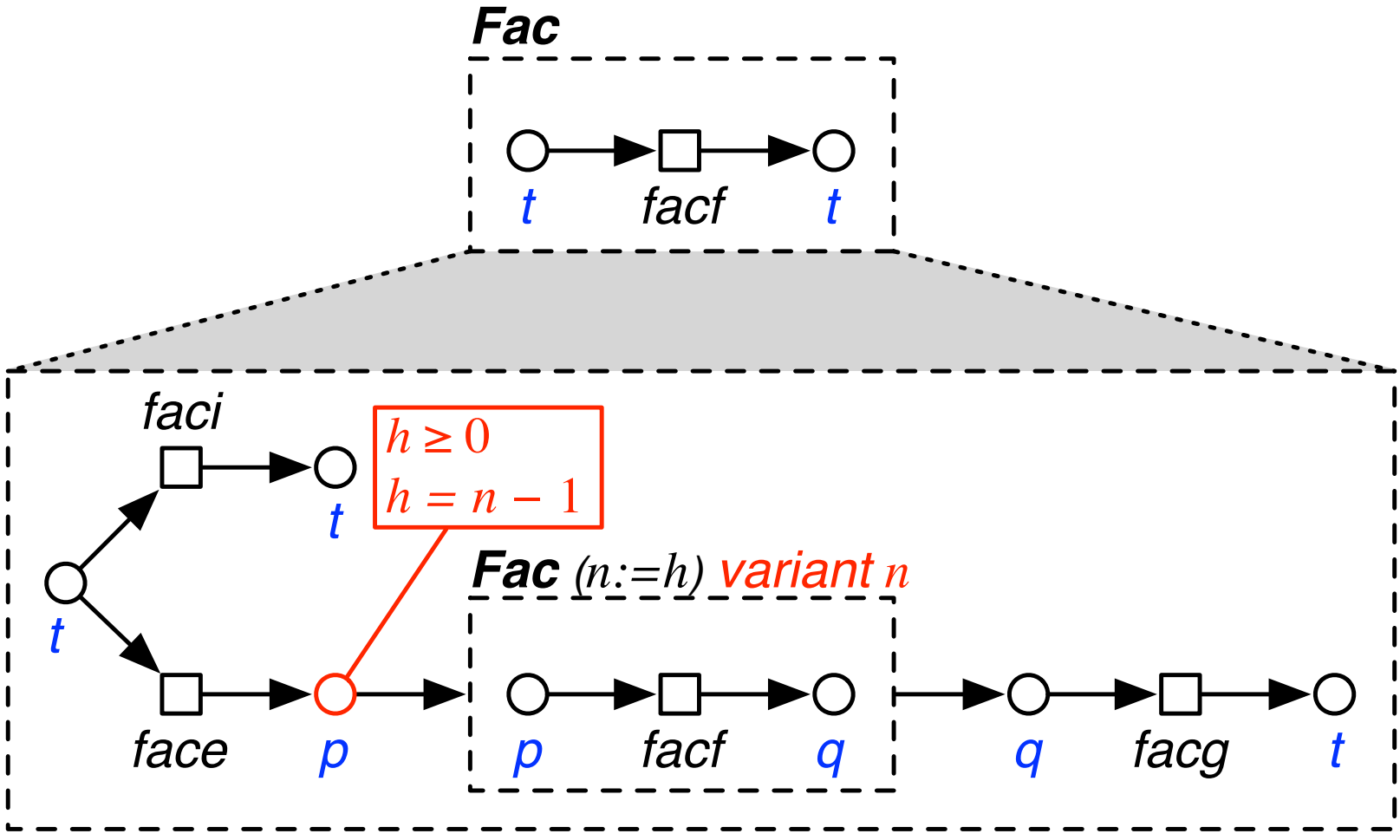
Termination of the Factorial Procedure



Termination of the Factorial Procedure



Termination of the Factorial Procedure



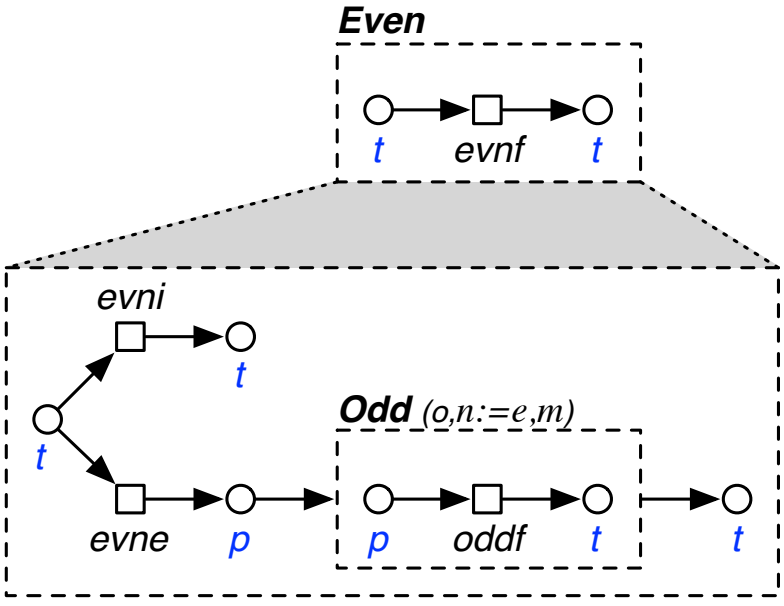
To be proved: $n \geq 0 \wedge n - 1 < n$

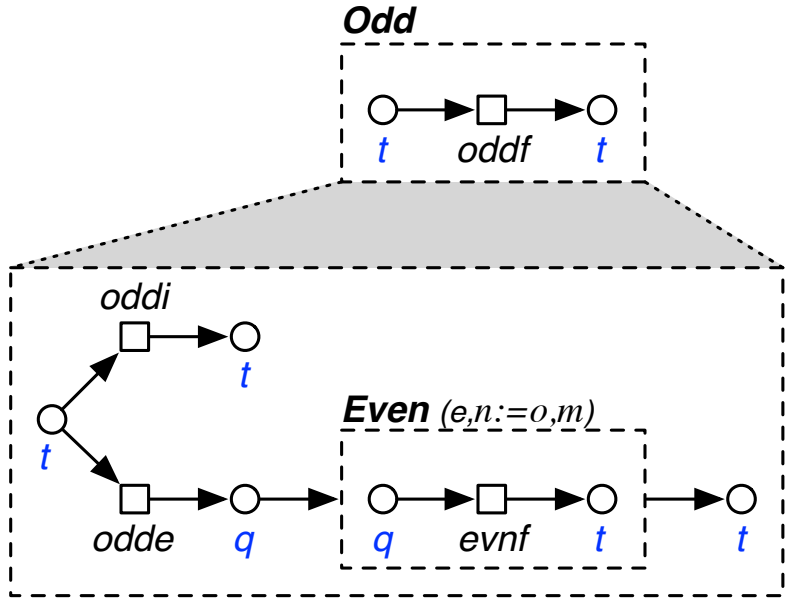
Contents

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Specification of EvenOdd

"Defunctionalisation" (J. C. Reynolds)





evnf = when $n \geq 0$ then $e :=$ "$n \bmod 2 = 0$" end

evni = when $n = 0$ then $e :=$ TRUE end
evne = when $n > 0$ then $m := n-1$ end

@*p* $m = n-1$

oddf = when $n \geq 0$ then $o :=$ "$n \bmod 2 \neq 0$" end

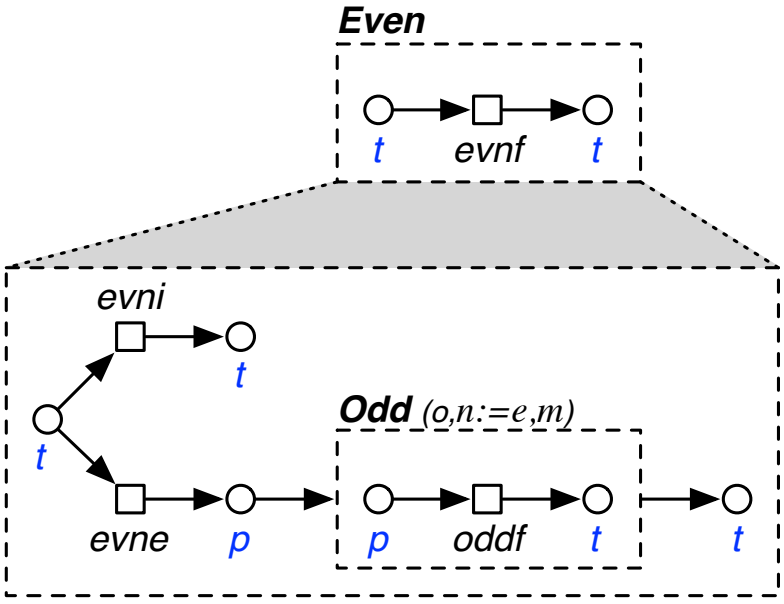
oddi = when $n = 0$ then $o :=$ FALSE end
odde = when $n > 0$ then $m := n-1$ end

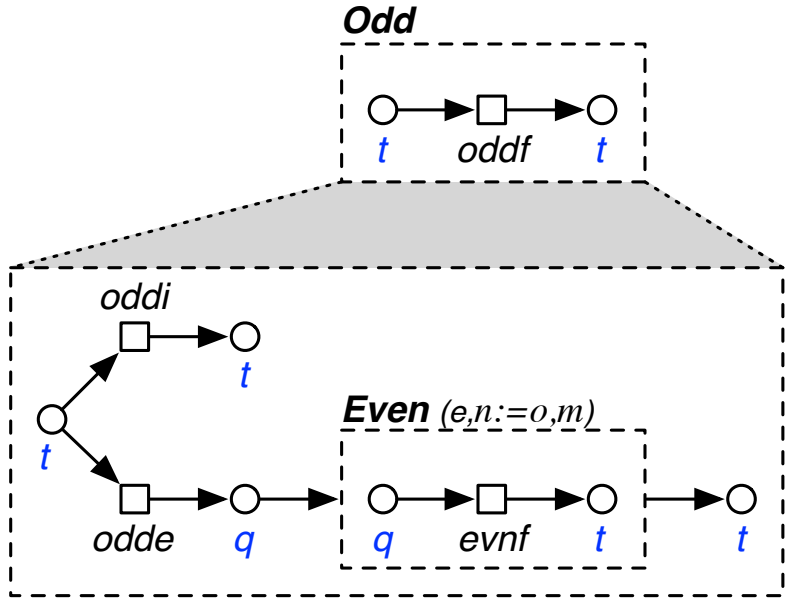
@*q* $m = n-1$

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Specification of EvenOdd

"Defunctionalisation" (J. C. Reynolds)





Even(e, n) ::
  if $n = 0$ then
    e := TRUE
  else
    var m := $n-1$; *Odd*(e, m)
  end

Odd(o, n) ::
  if $n = 0$ then
    o := FALSE
  else
    var m := $n-1$; *Even*(o, m)
  end

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Contents

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Conclusion

- Method for sequential program development
  - Recursion
  - Mutual Recursion
  - Termination
- To be investigated:
  - Modularity
  - Concurrency
  - Soundness