Automated Roundtrip Engineering

in Static Aspect Weaving and Transformations

Christoph Kessler

IDA / PELAB
Linköpings universitet, Sweden

Joint work with Mikhail Chalabine

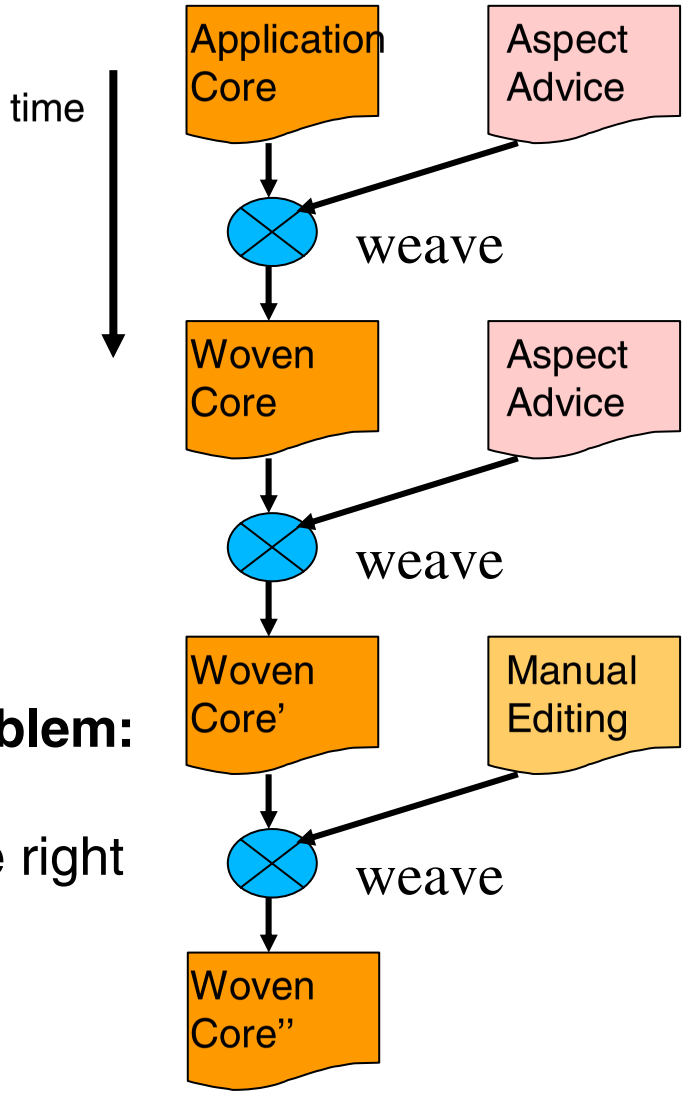
Proc. ICSE-2007 Minneapolis

Research funded by SSF RISE-PARACOMP,
CUGS Graduate School, LiU Ceniit

Static Aspect Weaving
... and other program transformations

- Static Aspect Weaving
- C macro expansion
- Rule-based source-to-source transformation
- Interactive (manual) editing

→ Sequences of (syntactic) program transformations (*weaving steps*)

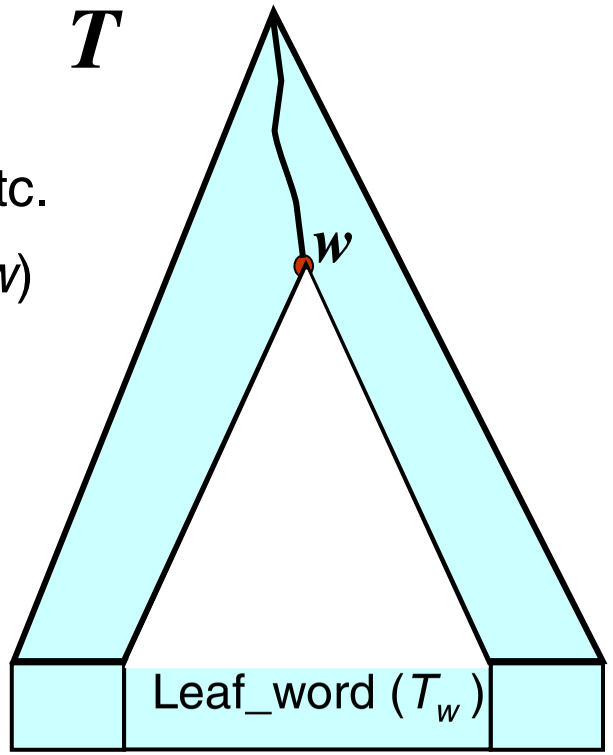


ARE (Automatic Roundtrip Engineering) Problem:
How to trace back and consistently commit (manual) changes in transformed code to the right source of origin?

- WovenCode → Core source
- WovenCode → Aspect source

...

Formalisation:
Concrete Syntax Tree (CST)



CST: **Parse tree / Derivation tree**

- Including whitespaces, semicolons, comments etc.
- Inner nodes *w* annotated with nonterminal: NT(*w*)
- Leaf node *b* annotated with terminal: NT(*b*)
- Inner node *w* with children $v_1,\ldots,v_k$:

  Production NT(*w*) → NT(v_1)…NT(v_k)

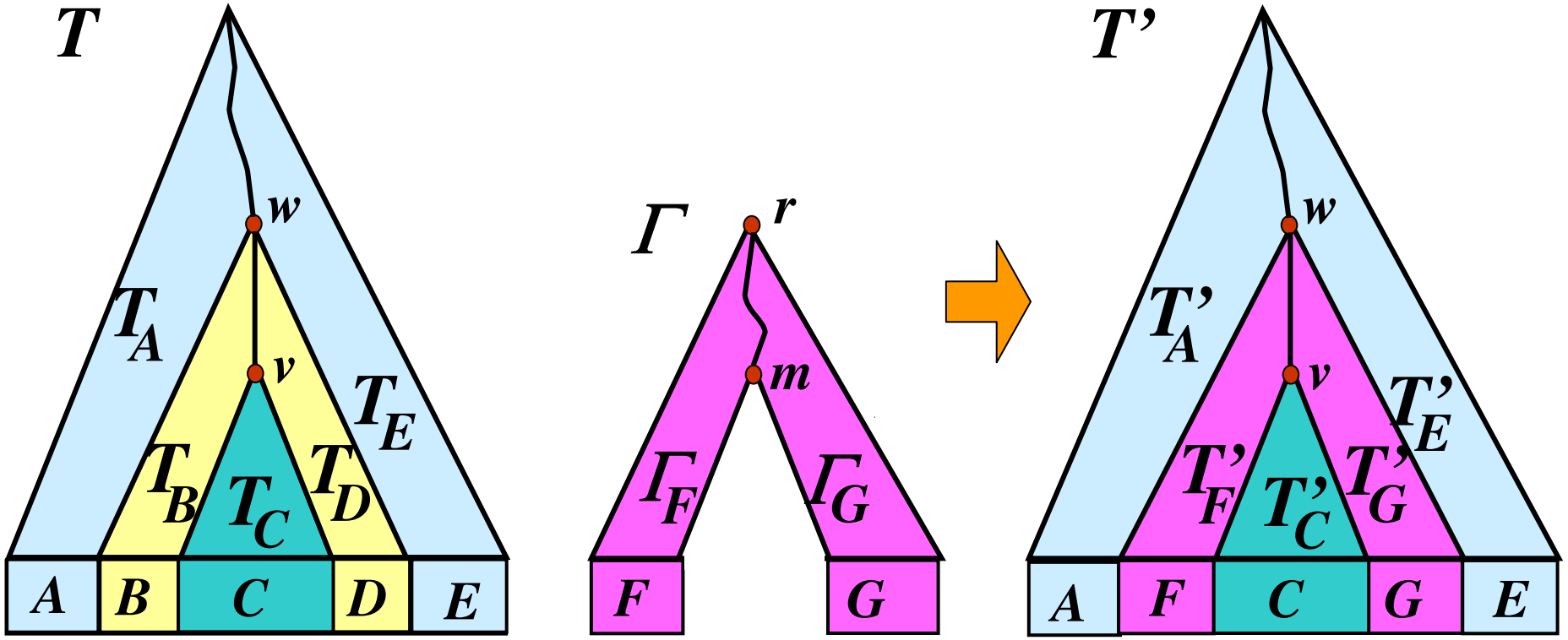
Syntactic Type of CST:
NT(root)

Generic CST / tree fragment: CST with ”open” leaves
(= non-expanded nonterminals)

Syntactic substitutability of T_w
only with tree (fragment) of type compatible to NT(*w*)

Formalisation (2)
Double-Grafting Transformation on CSTs

- **(1,1)-Double-Grafting-Transformation:**

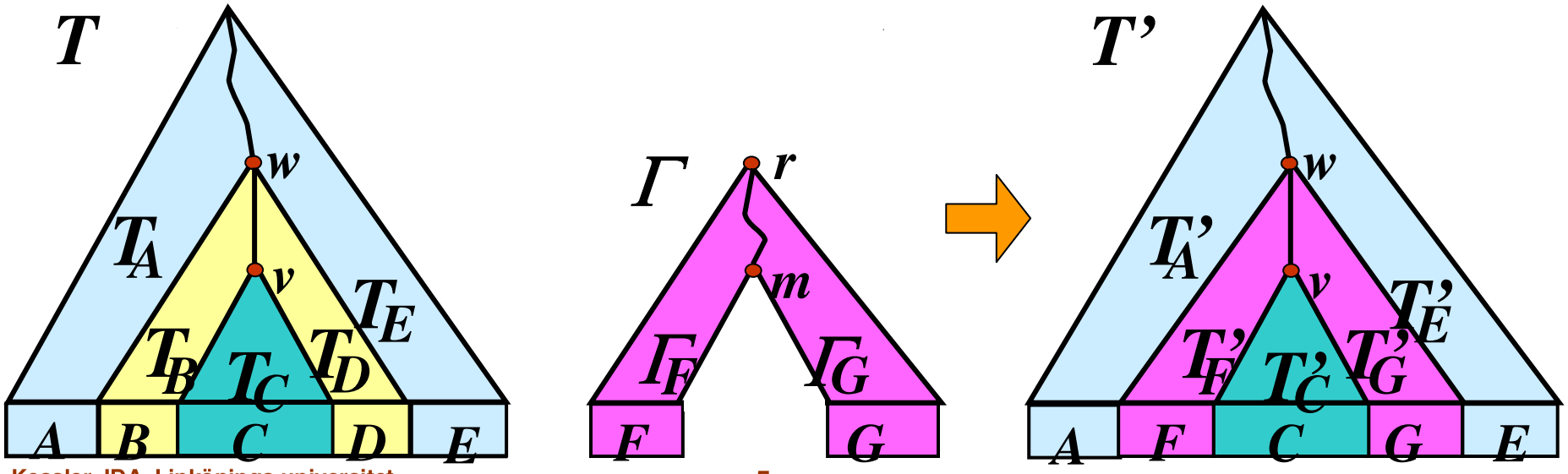


Notation: $T \vdash^{\Gamma} T'$

Syntactically correct
if NT(*w*) = NT(*r*)
and NT(*v*) = NT(*m*)

Example 1: Static Aspect Weaving

- Static Aspects (*before*, *after*, *around* join point) (e.g. early version of Aspect-J)
  - Application of an *advice* at a *join point (shadow)* is a (1,1)-Double-Grafting-Transformation
  - With missing *proceed*()-pseudocall (= no *m*-node): (1,0)-Double-Grafting Transformation



AOP Example

File Edit Window

Core:

```
1 public class BankTransaction {
2
3 public static void main(
4                     String[] args){
5   BankAccount ac =
6       new BankAccount(500, 400);
7   ac.credit(200);
8   ac.credit(100);
9 }
10}
11
12
```

Aspect:

```
public aspect secureTrans {

pointcut checkings(BankAccount bk):
    execution(*
        BankAccount.credit(float))
        && target(bk);

before(BankAccount bk) : checkings(bk) {
    assert bk.getBalance() > 250;
}
after(BankAccount bk) returning:
                            checkings(bk) {
    assert bk.getBalance() > 0;
}
}
```

Woven code:

```
5    BankAccount ac =
6       new BankAccount(500, 400);
7    assert ac.getBalance() > 250;
8    ac.credit(200);
9    assert ac.getBalance() > 0;
10   assert ac.getBalance() > 250;
11   ac.credit(200);
12   assert ac.getBalance() > 0;
```

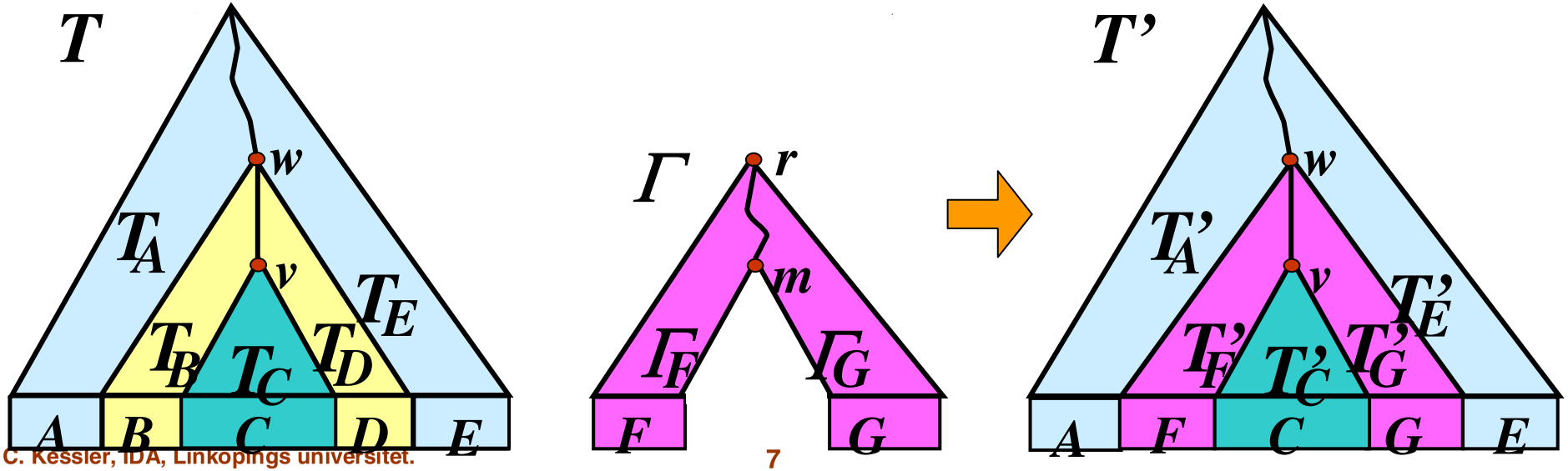
The woven code is in Aspect-J not visible for the programmer! (Weaving is done at bytecode level …)

Example 2: Editing

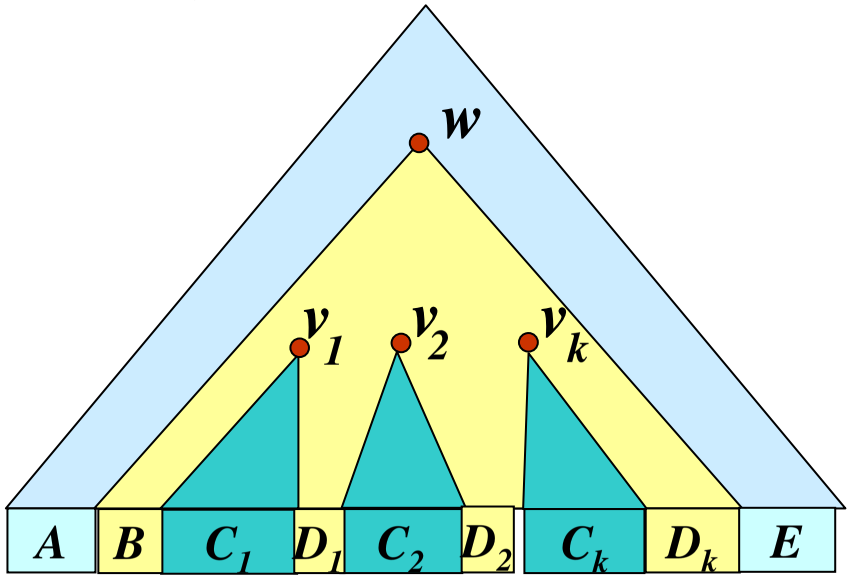
- Insert: (1,1)-Double-Grafting-Transformation
- Delete, replace: (1,0)-Double-Grafting-Transformation

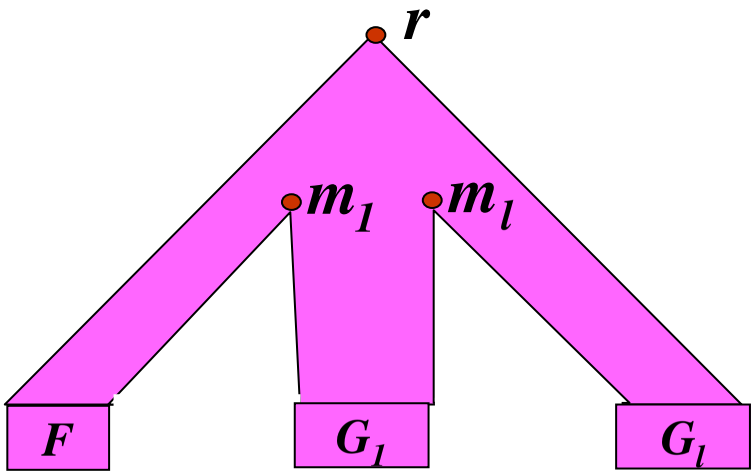
Requirements:

- Program again syntactically correct after editing
- Editor coupled with CST search structure –
  Clicking / Cursor on program point locates CST-node



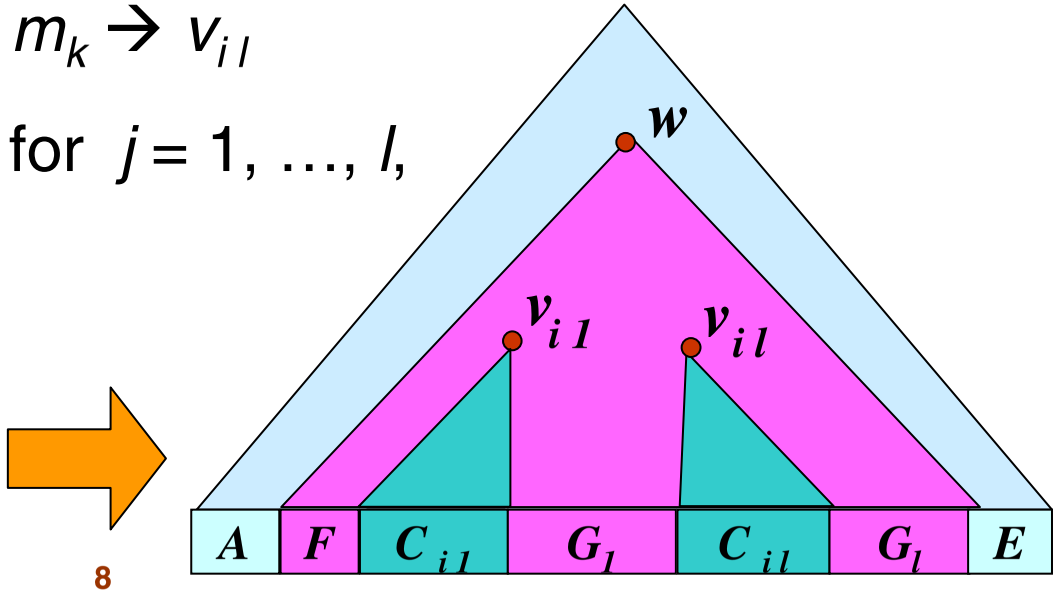
(*k, l*)-Double-Grafting-Transformation





+ Mapping: $m_1 \rightarrow v_{i\,1}$, …, $m_k \rightarrow v_{i\,l}$

where $v_{i\,j}$ in $\{ v_1, \ldots, v_k \}$ for $j = 1, \ldots, l$,
and $\mathrm{NT}(m_j) = \mathrm{NT}(v_{i\,j})$



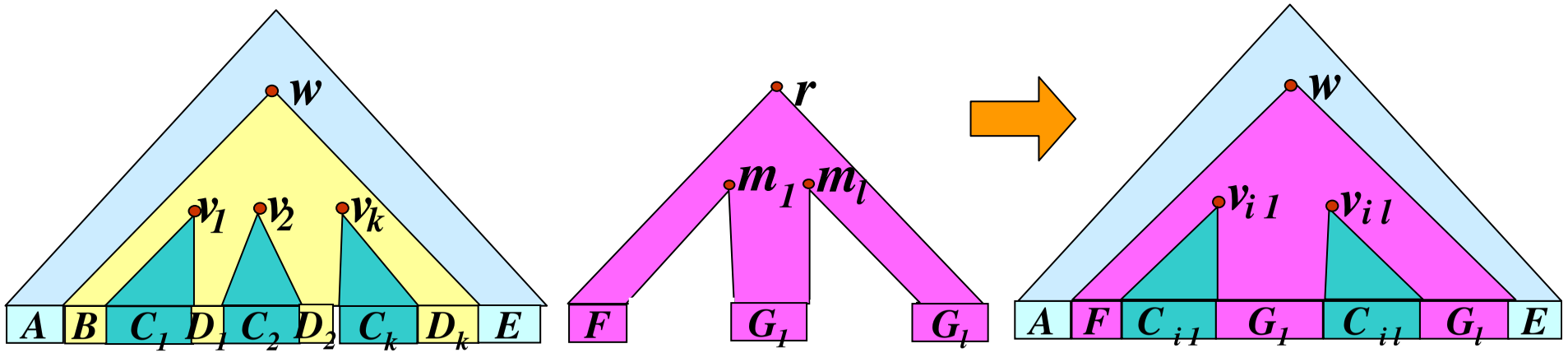
Example 3: C - Preprocessor

- Macro Expansion

```
#define  MAX( X, Y )   ( (X) > (Y)? X : Y )

… = MAX( u+1, 23 );
```

is a (2,4)-Double-Grafting-Transformation

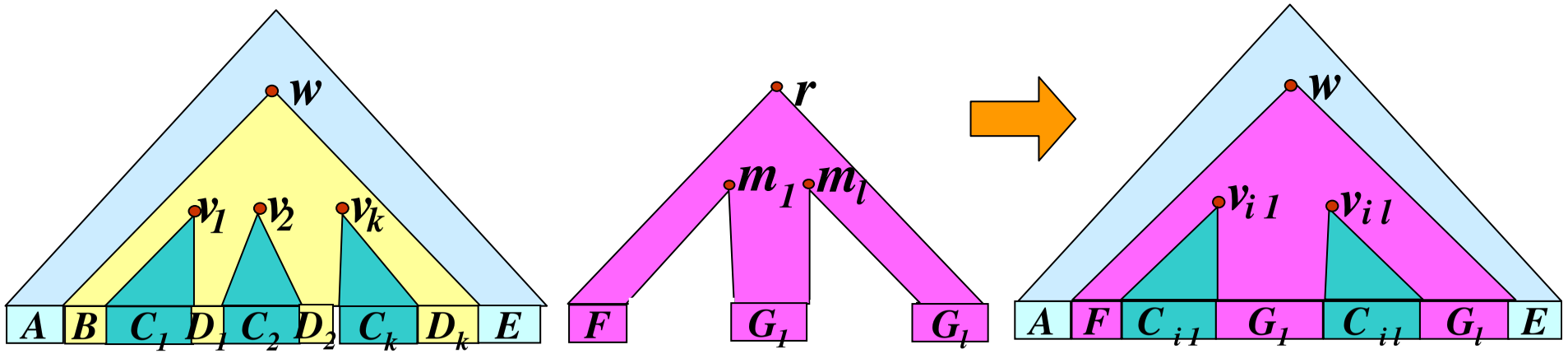


Example 4: Rule-based Source-to-Source Transformation Systems

- Generic Transformation Rule

```
@expr: @id:1 + @id:1   →   2 * id:1
```

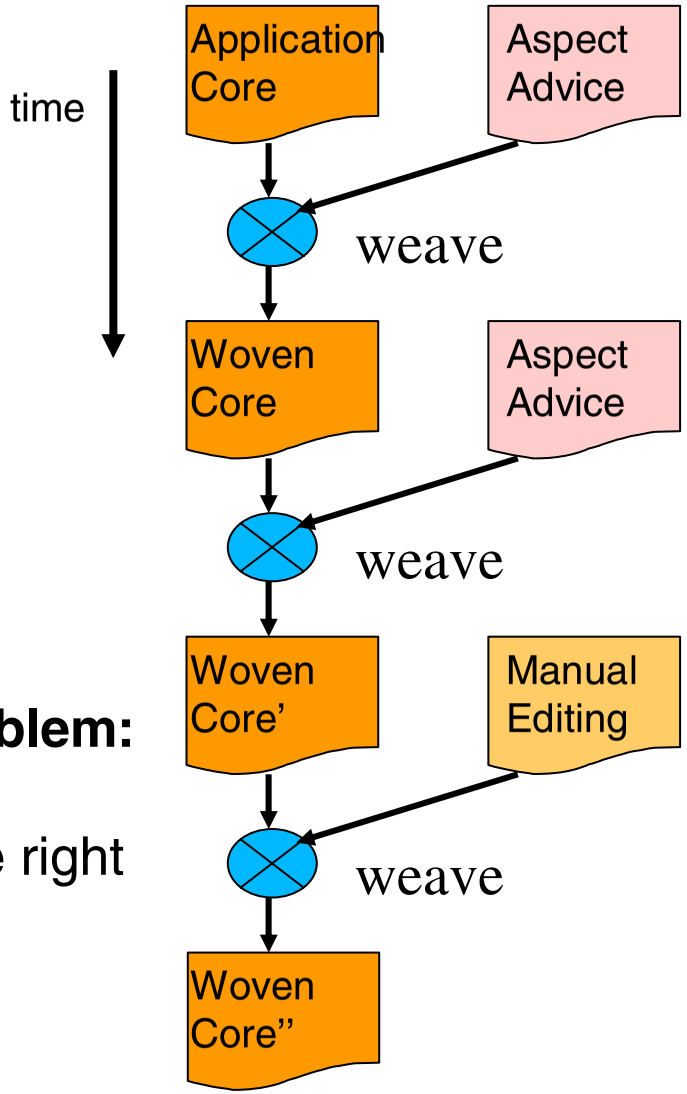
is a (2,1)-Double-Grafting-Transformation



Static Aspect Weaving
... and other program transformations

- Static Aspect Weaving
- C macro expansion
- Rule-based source-to-source transformation
- Interactive (manual) editing

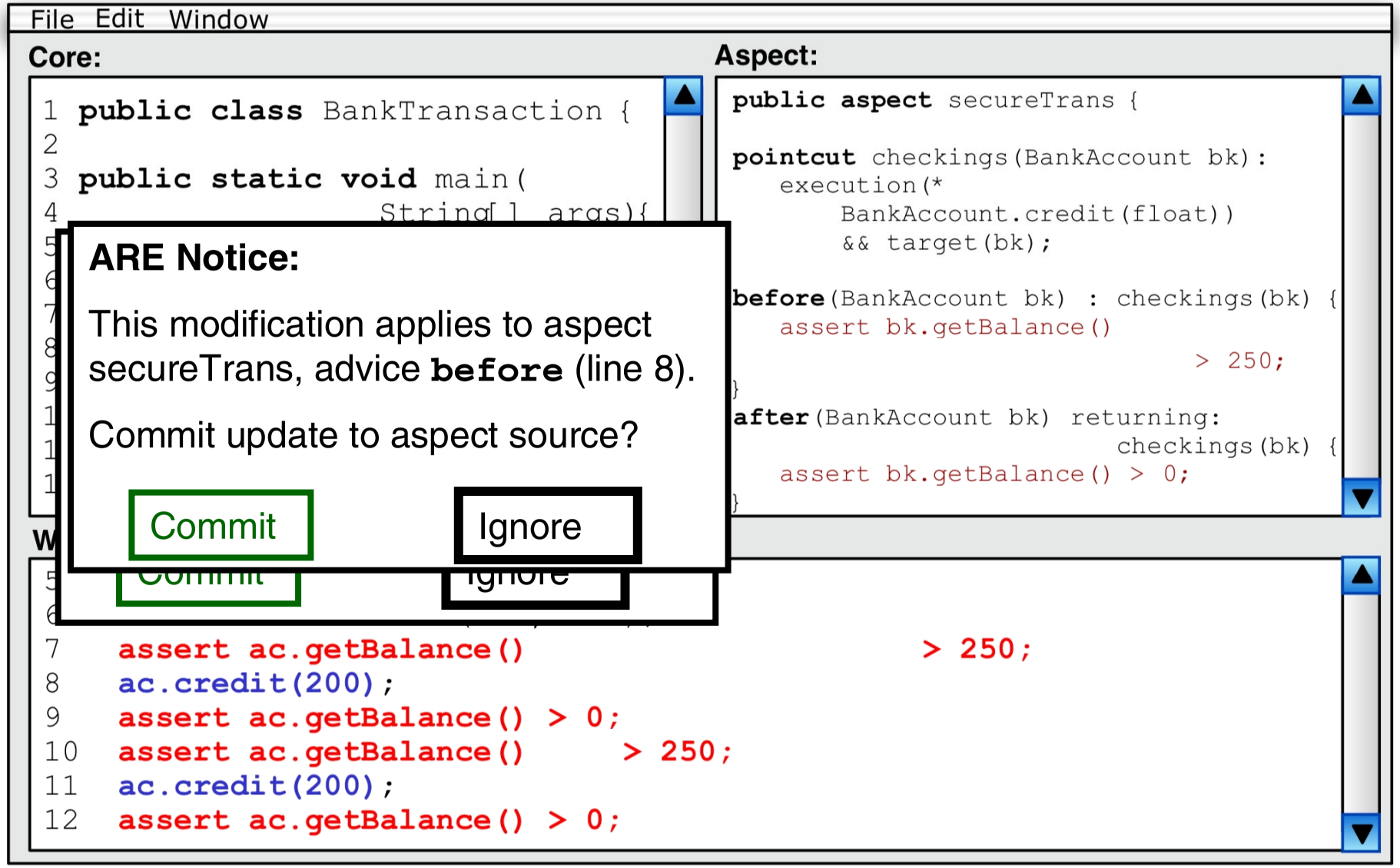
→ Sequences of (syntactic) program transformations (*weaving steps*)



ARE (Automatic Roundtrip Engineering) Problem:
How to trace back and consistently commit (manual) changes in transformed code to the right source of origin?

- WovenCode → Core source
- WovenCode → Aspect source

ARE Example


File Edit Window
Core:
1 public class BankTransaction {
2
3 public static void main(
4 String[] args){
Aspect:
public aspect secureTrans {
pointcut checkings(BankAccount bk):
execution(*
BankAccount.credit(float))
&& target(bk);
before(BankAccount bk) : checkings(bk) {
assert bk.getBalance()
> 250;
}
after(BankAccount bk) returning:
checkings(bk) {
assert bk.getBalance() > 0;
}
ARE Notice:
This modification applies to aspect secureTrans, advice before (line 8).
Commit update to aspect source?
Commit
Ignore
7 assert ac.getBalance()
> 250;
8 ac.credit(200);
9 assert ac.getBalance() > 0;
10 assert ac.getBalance()
> 250;
11 ac.credit(200);
12 assert ac.getBalance() > 0;

Problem:
Find Origin of a Program Point

- Given: Weaving sequence (History)

$$T_0 \vdash^{\Gamma_1} T_1 \vdash^{\Gamma_2} T_2 \vdash^{\Gamma_3} T_3 \ldots \vdash^{\Gamma_n} T_n$$

- Given: Program point coordinate w.r.t. T_n
  (row *r*, column *c*) in Editor
  or, linear index *p* in Lexeme Sequence (leaf word)

- Compute: Source-File + Coordinate of *p* in Origin of *p*

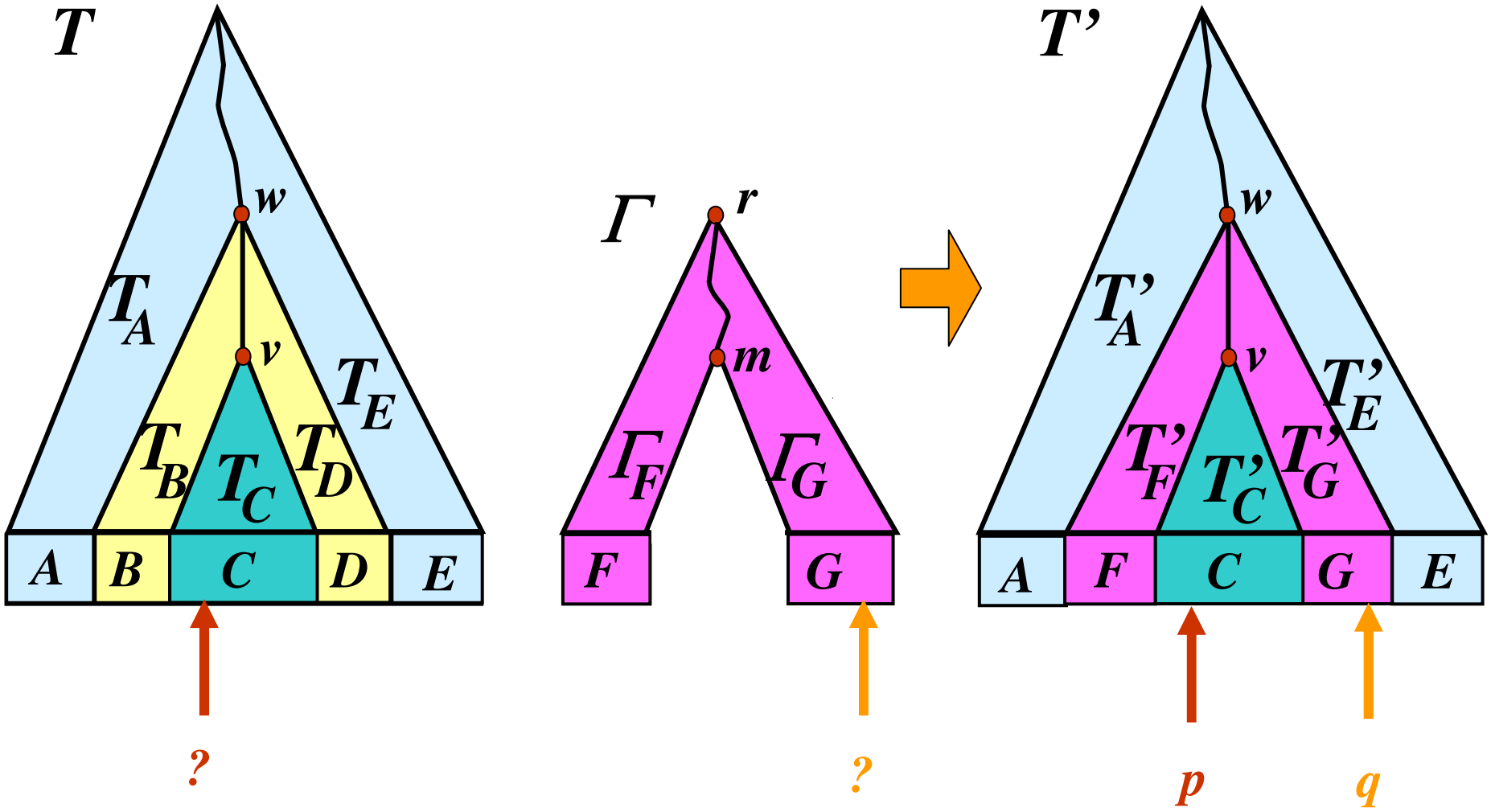
  either in the core T_0
  or in one of the Γ_i, $i = 1 \ldots n$

- Method: Store relative coordinate transformation to each Double-Grafting-Transformation in the weaving sequence

  → Transposition tree

Transposition Tree Construction (1)

- As example: for a (1,1)-Double-Grafting-Transformation



Transposition Tree Construction (2)

Forward-Transposition:

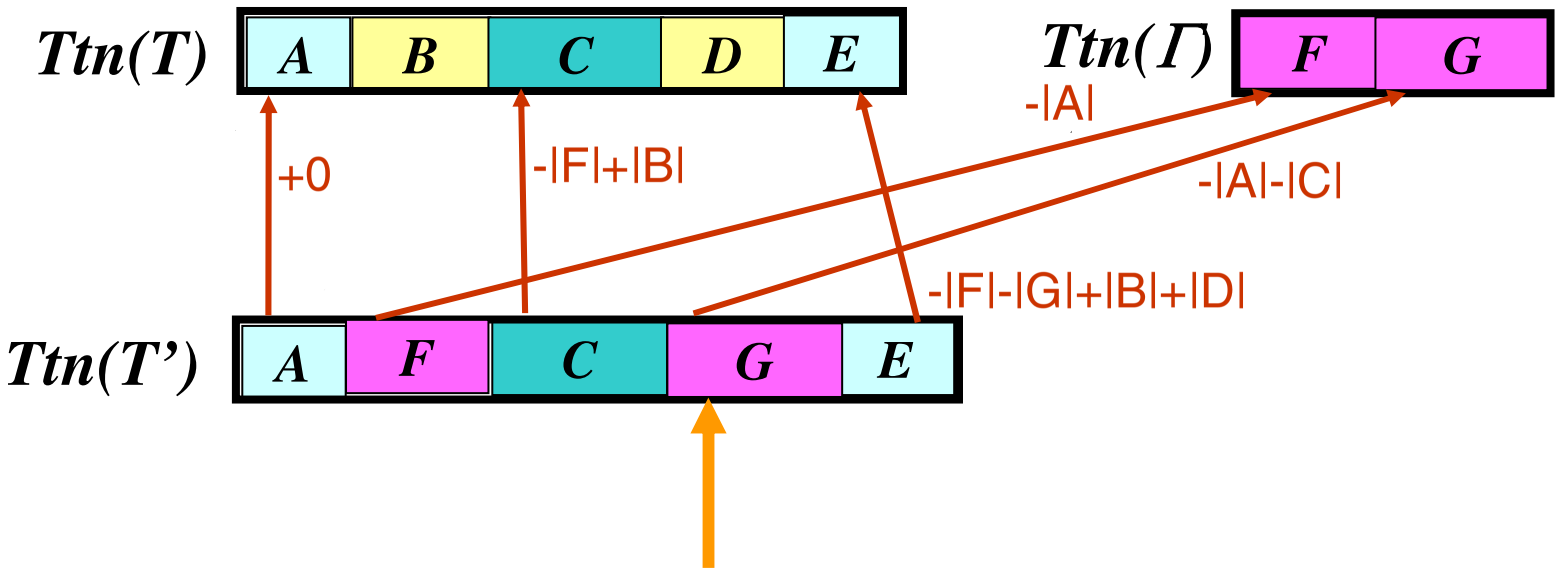
$$fsrc_{T'}(v) = \begin{cases} fsrc_T(v) & \text{if } v \text{ in } T_A, T_C, T_E \\ fsrc_\Gamma(v) & \text{if } v \text{ in } T_F, T_G \\ \text{undefined,} & \text{if } v \text{ in } T_B, T_D \end{cases}$$

$$fpos_{T'}(v) = \begin{cases} fpos_T(v) & \text{if } v \text{ in } T'_A \\ fpos_\Gamma(v) + |A| & \text{if } v \text{ in } T'_F \\ fpos_T(v) + |A| - |B| + |F| & \text{if } v \text{ in } T'_C \\ fpos_\Gamma(v) + |A| - |B| + |F| + |C| & \text{if } v \text{ in } T'_G \\ fpos_T(v) + |A| - |B| + |F| + |C| - |D| + |G| & \text{if } v \text{ in } T'_E \end{cases}$$

Transposition Tree Construction (3)

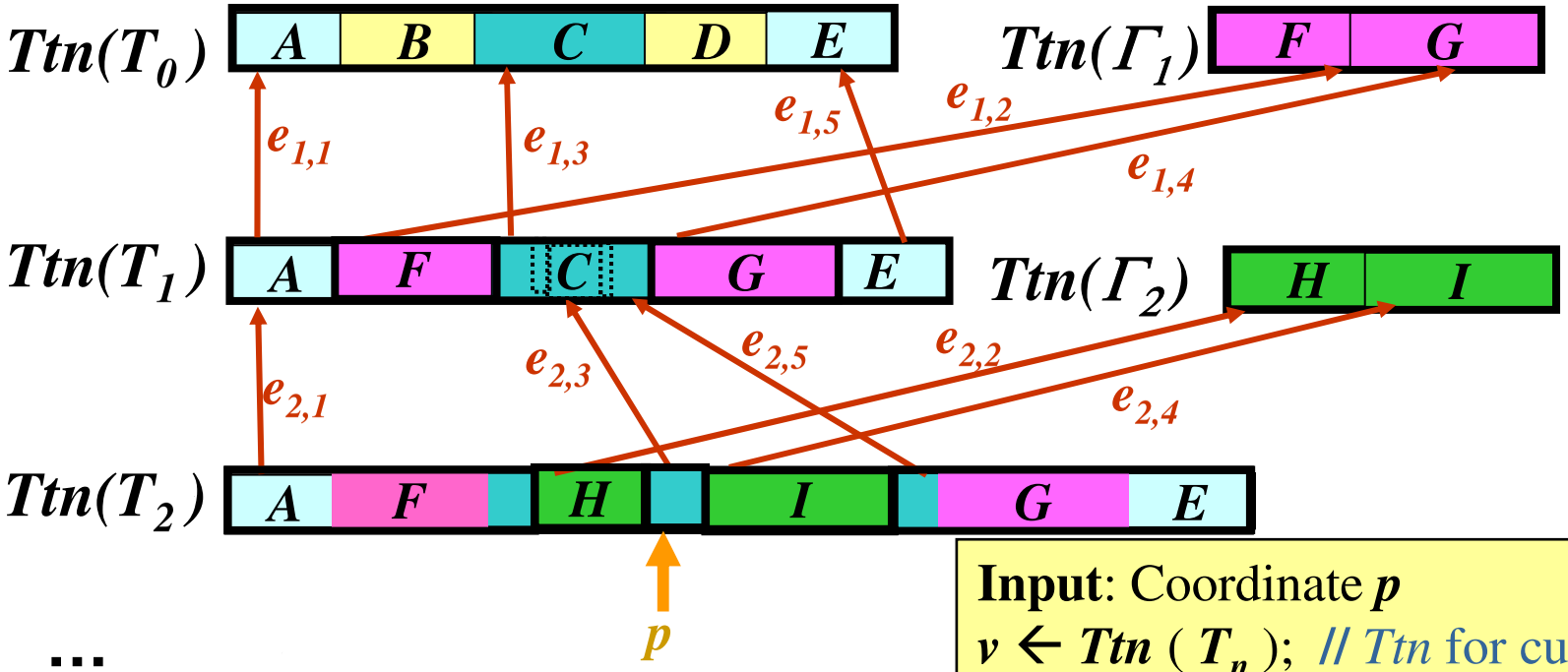
- All coordinates in the same segment (*A, B*, …) are mapped (transposed) in the same way
- Backward transposition stored blockwise

  → Nodes / Edges in Transposition tree



Transposition Tree Construction (4)

- Incremental construction yields tree of depth <= *n*



Data structure needs O(*n*) space (Constant depends on *k*, *l*)

Search backwards in time O(*n*)

Input: Coordinate ***p***
v ← ***Ttn*** (T_n); *// Ttn for current CST*
for ***t*** **from** ***n*** **downto** 1 **do**
  find interval ***i*** of ***p*** in ***v*** (Binary search)
  p ← Backwardtransp[$e_{t,i}$] (***p***);
  v ← target ($e_{t,i}$);
  if ***v*** is *Ttn* of an aspect ***Γ*** **break**
od

Propagating Changes (1)

- Weaving sequenz can always be *replayed*
  - Undo Γ_n by replaying $\Gamma_1,\ldots,\Gamma_{n-1}$ on T_0

At a change (e.g. edit):

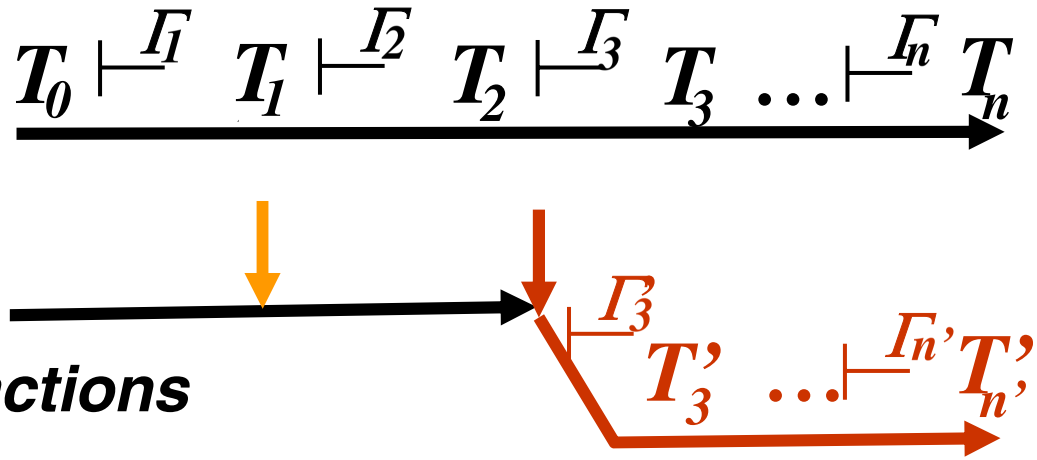
- Origin in core T_0 → 2 Options (User dialog):
  - Store and commit edit as local patch Γ_{n+1} of T_n
  - Patch edit directly in the core (as Γ_0), then replay $\Gamma_1,\ldots,\Gamma_n$

- Origin in Aspect-Advice → 3 Options (User dialog):
  - Store and commit edit as local patch Γ_{n+1}
  - Patch edit directly in Advice (as Γ_0), then replay $\Gamma_1,\ldots,\Gamma_n$
  - Commit after Γ_t for some $0<t<n$, then replay $\Gamma_{t+1},\ldots,\Gamma_n$

Propagating Changes (2)

Problems:

- Not all T_t are (semantically) correct programs
- Paradox (cf. ”Back to the future II”)
  - Change can lead to disruptive changes in the replay – code segments created ”later” may disappear, new ones appear…



Remedy: ***Weaving Transactions***

- Encapsulate inconsistent intermediate configurations
- Limit scope of propagating changes, undo option

Application?

- Planned: Interactive Parallelisation Tool
  - Rule-based automatic transformations
  - Semiautomatic transformations (Skeleton expansion)
  - Manual parallelisation (forall, sh, $ivdep, $distribute …)
  - Sequential application core remains intact
  - Co-Evolution of application core and woven-in parallelisation code

Summary

- (*k, l*)-Double-Grafting-Transformation
  - Models static aspect weaving, Macro expansion, Edits …
- Store weaving sequence
  - with transposition tree
    → find origin of a program point
  - Replay
  - User dialog for consistent change propagation

☺ Need not know reverse transformation to each step

☺ Space-efficient data structure

References

- Uwe Assmann, Andreas Ludwig:
  Aspect Weaving by Graph Rewriting.
  Proc. Generative Component-Based Software Engineering (GCSE), 1999

- Uwe Assmann:
  Automatic Roundtrip Engineering.
  Electronic Lecture Notes in Theoretical Computer Science **85**(5), 2003

- Mikhail Chalabine, Christoph Kessler, Peter Bunus:
  Automated Round-trip Software Engineering in Aspect Weaving Systems (Short paper),
  Proc. of the 21st IEEE / ACM International Conference on Automated Software Engineering (ASE 2006), Tokyo, Japan, September 2006

- Mikhail Chalabine, Christoph Kessler:
  A Formal Framework for Automated Round-trip Software Engineering in Static Aspect Weaving and Transformations,
  Proc. of the 29th IEEE / ACM SIGSOFT Int. Conference on Software Engineering (ICSE 2007), Minneapolis, MN, USA, May 2007