COMBINING FEATURE ANNOTATIONS AND SUPERIMPOSITION IN FORMAL MODELING

Philipp Chrszon

Institute for Software Technology, German Aerospace Center (DLR), Braunschweig

FOSD Meeting 2024

DLR

Annotative and Compositional Approaches
for implementing features

Annotative

- **Mark fragments** of feature-specific code
- Derive variant using **projection**
- Examples:
  - `#ifdef`
  - Featured Transition Systems
- Arbitrary **granularity**, construction of **family-model** straightforward
- Modularization difficult, poor traceability

Compositional

- Implement features as distinct **modules**
- Create variant using **composition**
- Examples:
  - Jak
  - fSMV
- Feature **modularity**, easy separation of concerns
- Limited granularity, construction of family model challenging

Goal: Combine **annotative** and **compositional** approaches in **formal modeling** to enable:

- **Fine-grained** modifications
- Feature **modularization**
- **Family-based analysis** and dynamic reconfiguration

Formal Behavioral Modeling
using guarded commands

$$[action]\ f\text{-}guard \wedge guard \rightarrow update$$

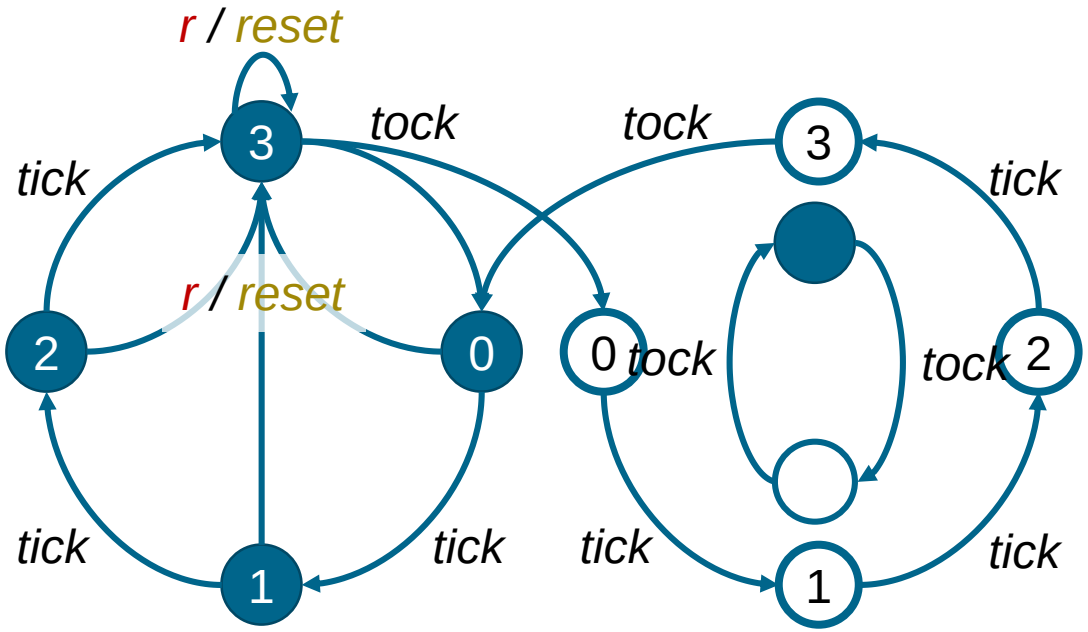
- Boolean condition over features
- Boolean condition over variables
- Variable updates

Example: 2-bit counter

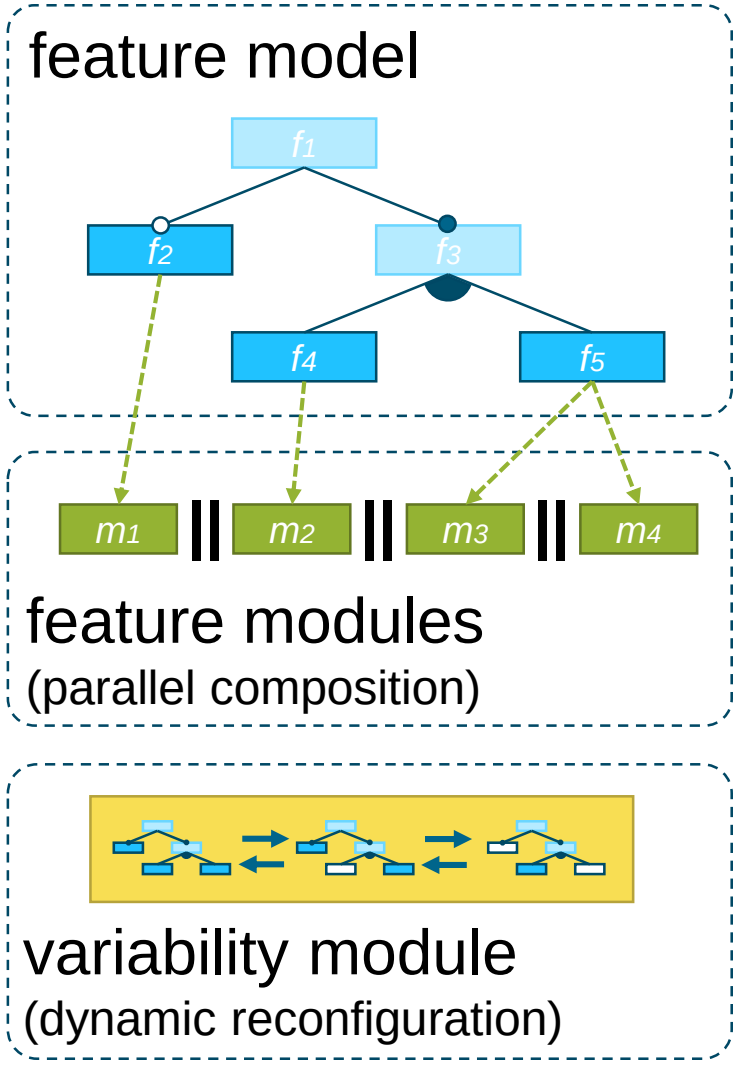
P_1:
[*tick*] $(c < 3) \rightarrow (c' = c + 1)$
[*tock*] $(c = 3) \rightarrow (c' = 0)$
[*reset*] $r \wedge tt \rightarrow (c' = 0)$

P_2:
[*tock*] $tt \rightarrow (b' = \neg b)$

$P_1 \parallel P_2$:
[*tick*] $(c < 3) \rightarrow (c' = c + 1)$
[*tock*] $tt \wedge (c = 3) \rightarrow (c' = 0) \wedge (b' = \neg b)$



The ProFeat Language
annotative feature-oriented modeling




```
[step] (active(executive) & num_floorreq(top)>0
  ? ( cabin_pos=top ? 1 : 0)
  : ( active(ttf) & (load/size > 2/3)
      ? num_cabinreq(k)
      : num_cabinreq(k) + num_floorreq(k)
    )) > 0 |
    !(active(overload) => (load <= size)) |
    !(active(parking) & idleing => !(cabin_pos=eg))
    -> (cabin_open' = true);
```

P. Chrszon, C. Dubslaff, S. Klüppelholz, C. Baier: “ProFeat: Feature-oriented Engineering for Family-based Probabilistic Model Checking”, Formal Aspects of Computing, Vol. 30, No. 1, pp. 45-74, 2018.

Composition by Superimposition
modifying guarded commands using delta programs

- **Superimposition** describes how a system or module is **modified** upon **composition**
- **Extension** by adding commands to module
- **Modification** and removal via modification function:

$$m : Cmd \rightarrow 2^{Cmd}$$

- Example: saturating counter

P_1:
[*tick*] $(c < 3) \rightarrow (c' = c + 1)$
[*tock*] $(c = 3) \rightarrow (c' = 0)$

Δ:
m(“[*tock*] $(c = 3) \rightarrow (c' = 0)$”) = “[*tick*] $(c = 3) \rightarrow$ true”

- **Combination** of superimposition and annotative approach by allowing annotations in feature modules and deltas

Clemens Dubslaff: “Compositional Feature-Oriented Systems“, Software Engineering and Formal Methods (SEFM), 2019.
Clemens Dubslaff: “Quantitative Analysis of Configurable and Reconfigurable Systems”, PhD thesis, TU Dresden, 2022.

Combining Feature Annotations and Superimposition in the ProFeat language

```
feature Elevator
  module cabin
    pos : [0..5] init 0;
    dir : {UP, DOWN} init UP;
    door : {CLOSED, OPEN} init CLOSED;

    [tick] door = CLOSED & dir = UP & req_above ->
      (pos' = pos + 1) & (dir' = if pos = 4 then DOWN else UP);
    [tick] door = CLOSED & dir = DOWN & req_below ->
      (pos' = pos - 1) & (dir' = if pos = 1 then UP else DOWN);
  endmodule
endfeature

feature Parking
  change module Elevator.cabin
     rewrite floor_buttons[0] to
     (if !active(Executive) & num_req_total = 0 then true else floor_buttons[0]);
  endmodule
endfeature
```


M. Plath, M. Ryan: “Feature Integration using a Feature Construct”, Sci. Comp. Program. 41-1, pp. 53-84, 2001.

Discussion
design decisions and open questions

- Unify or distinguish the **two kinds** of feature modules?
  1. Adding new parallel modules
  2. Modify existing parallel modules using superimposition
- Implicit or explicit **interfaces** (variables, actions) in delta modules?
- Superimposition is **order-dependent**, how is this resolved?
  - Derive from dependency graph?
  - Derive from feature model?
  - Explicit definition?
- Interaction between **rewrites and function calls**: Should rewrites enter function bodies?

Impressum

Thema:	**Combining Feature Annotations and Superimposition in Formal Modeling** FOSD Meeting 2024
Datum:	09.04.2024
Autor:	Philipp Chrszon
Institut:	Softwaretechnologie
Bildcredits:	