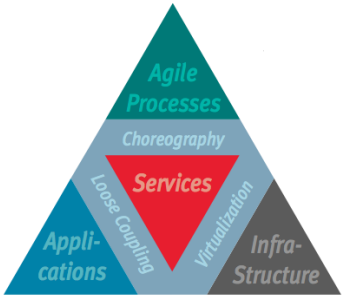
# From the How to the What

# Static Analysis via Model Checking

Bernhard Steffen
TU Dortmund



The difference between model checking and program analysis is this.:
When you call a model checker,
it runs and runs and never comes back;
when you do a program analysis,
it comes back immediately and says „don‘t know“.

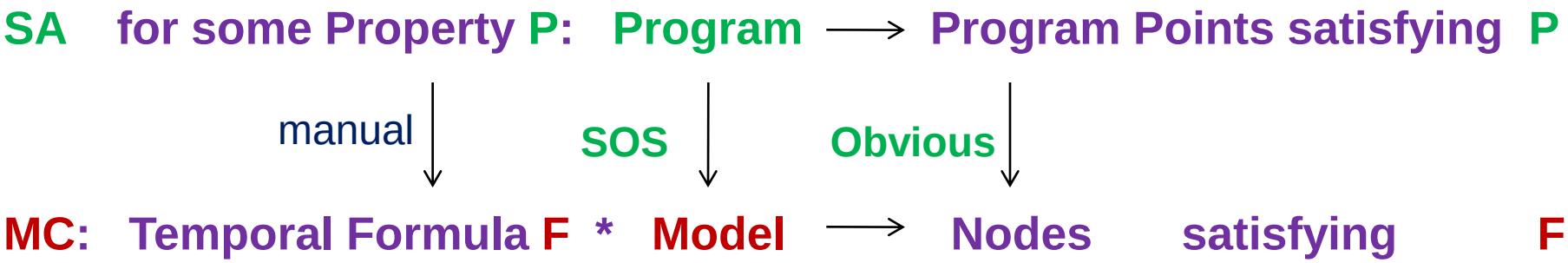
Patrick Cousot

Model checking is the type checking of tomorrow

# Playing the Association Game

| Static Analysis | Model Checking |
|---|---|
| Program/Data-Oriented | **Property-Oriented** |
| Algorithmic Specifications | **Temporal Specifications** |
| Analysis Frameworks | **(Generic) Model Checkers** |
| Practice-Driven | Theory-Driven |
| Complex Program Structures | **‚simple' Computational Structures** |
| Complex Data Structures | Abstract Entities |
| Structural Heuristics | Efficient Codings |
| Analysis | Verification |
| Missed Transformations | Some Diagnostics |
| Global Analysis | Local Analysis |

# SA Generation




**Generated analyses were more efficient than the handwritten counterparts!**

- Better structure
- Optimized Model Checkers (Fixpoint Analysis Machine)
- Dwyer and Robby example

# Outline

- Motivation
- **Control Flow (Generating Models)**
- Static Analysis as Model Checking
- Conclusions

# Adding Lables

$$[\mathrm{ass}_{SOS}] \quad \frac{}{\langle \mathbf{x} = a, \bullet \rangle \xrightarrow{\mathbf{x} = a} \bullet}$$

$$[\mathrm{skip}_{SOS}] \quad \frac{}{\langle \mathbf{skip}, \bullet \rangle \xrightarrow{skip} \bullet}$$

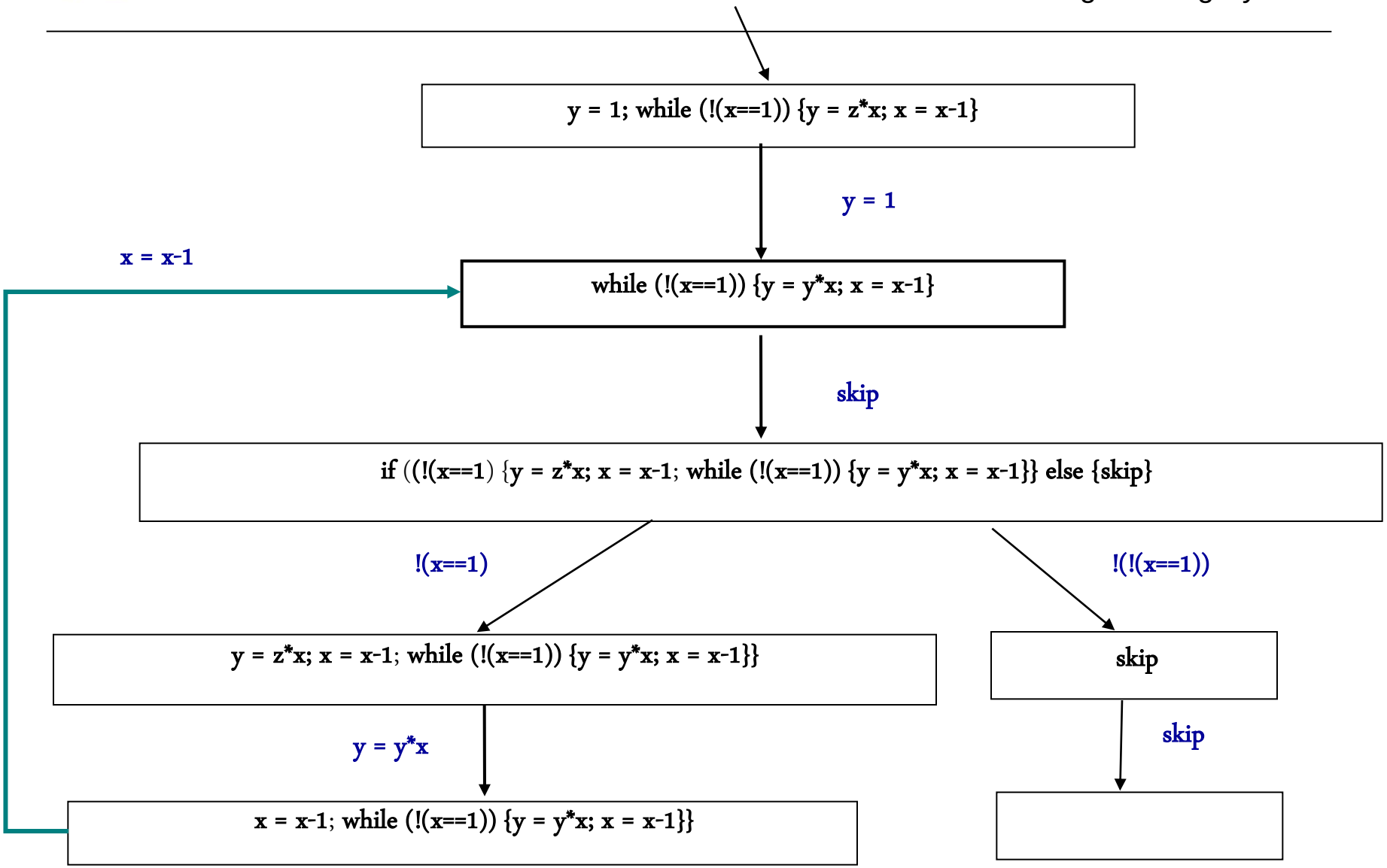
$$[\mathrm{comp}^1{}_{SOS}] \quad \frac{\langle S_1, \bullet \rangle \xrightarrow{\lambda} \langle S_1', \bullet \rangle}{\langle S_1; S_2, \bullet \rangle \xrightarrow{\lambda} \langle S_1'; S_2, \bullet \rangle}$$

$$[\mathrm{comp}^2{}_{SOS}] \quad \frac{\langle S_1, \bullet \rangle \xrightarrow{\lambda} \bullet}{\langle S_1; S_2, \bullet \rangle \xrightarrow{\lambda} \langle S_2, \bullet \rangle}$$

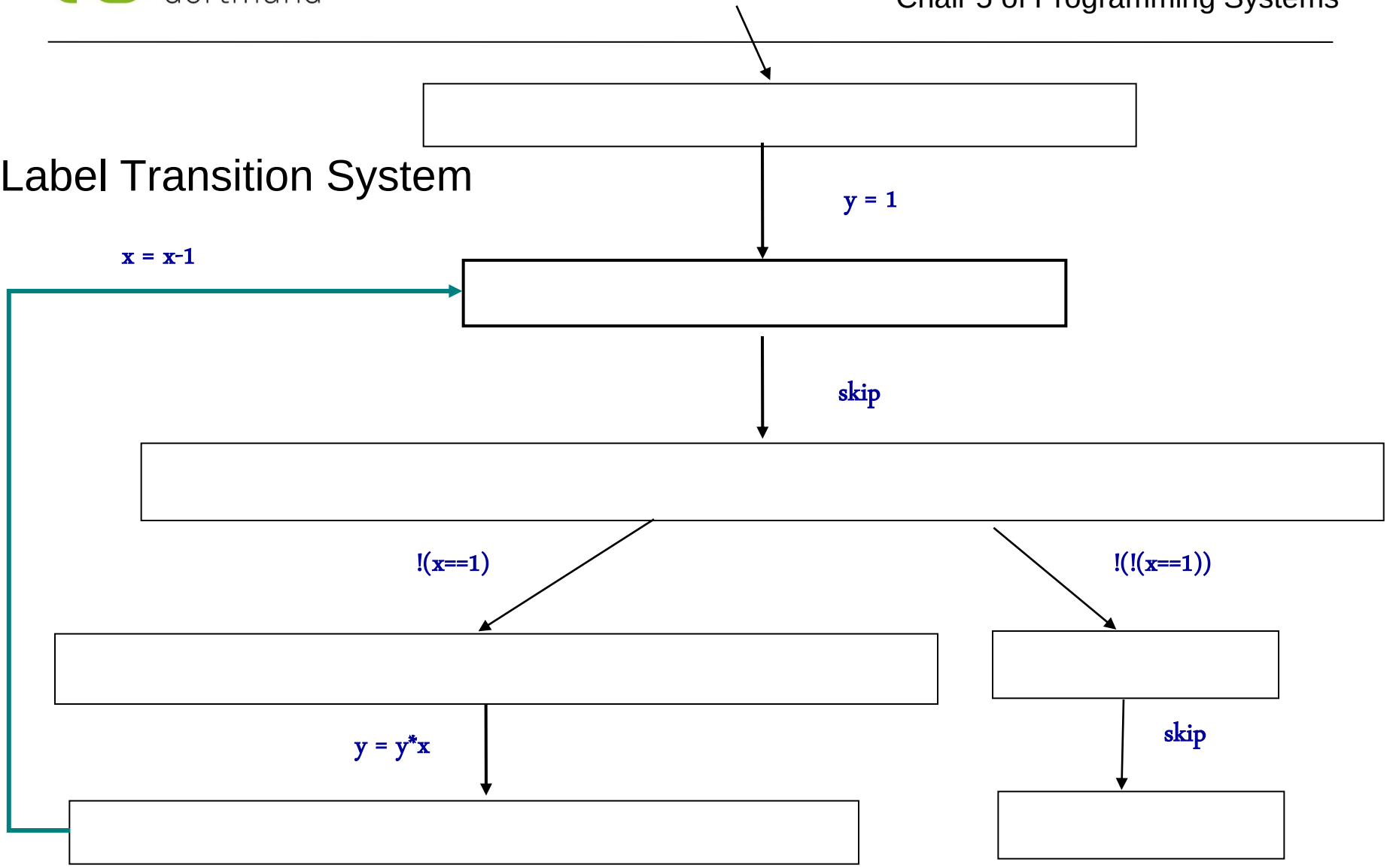
$$[\text{if}^{tt}_{SOS}]\quad \frac{}{\langle\ \textbf{if}\ (b)\ \{S_1\}\ \textbf{else}\ \{S_2\},\ \bullet\ \rangle \overset{b}{\Rightarrow} \langle\ S_1,\ \bullet\ \rangle}$$

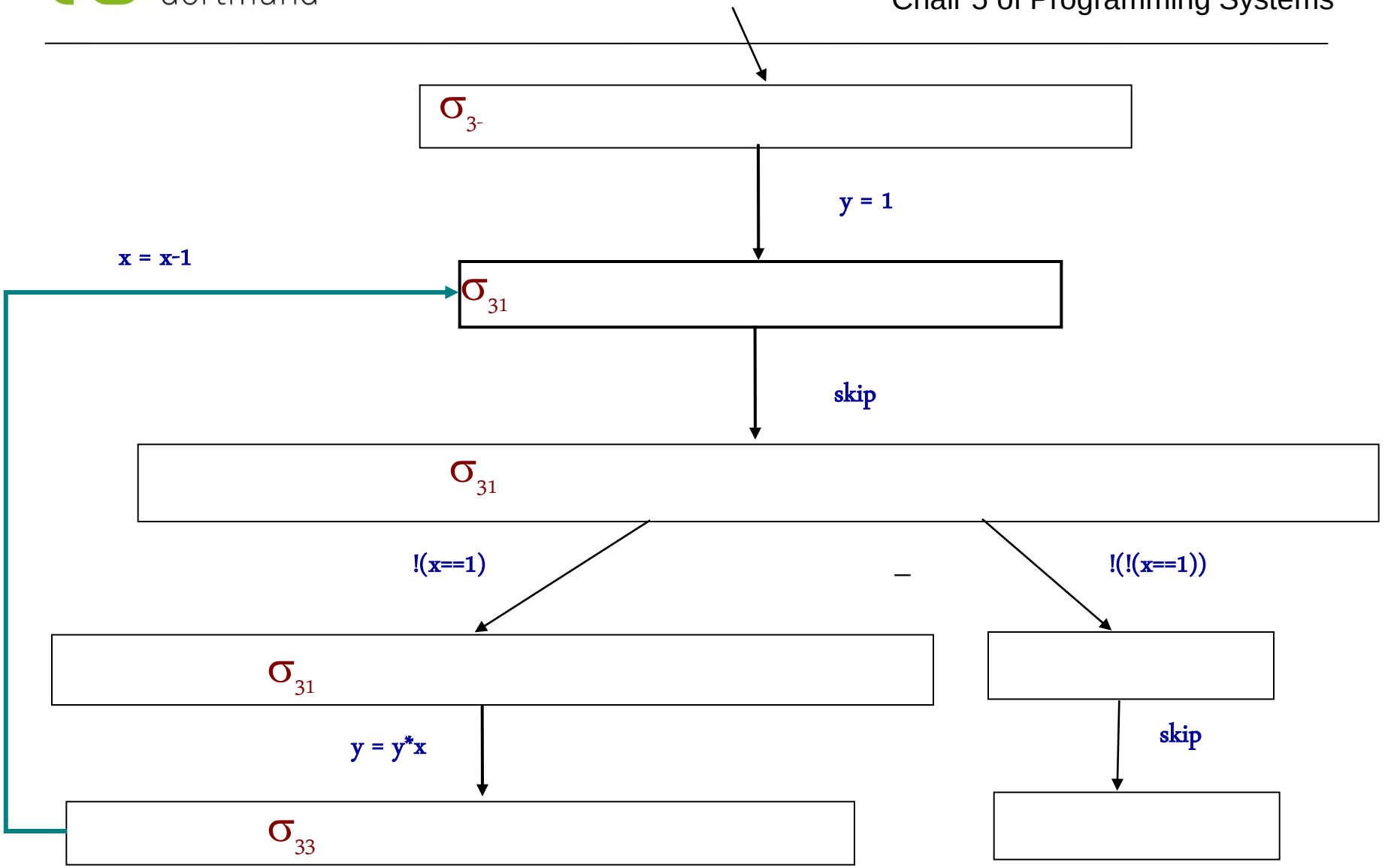
$$[\text{if}^{ff}_{SOS}]\quad \frac{}{\langle\ \textbf{if}\ (b)\ \{S_1\}\ \textbf{else}\ \{S_2\},\ \bullet\ \rangle \overset{!b}{\Rightarrow} \langle\ S_2,\ \bullet\ \rangle}$$

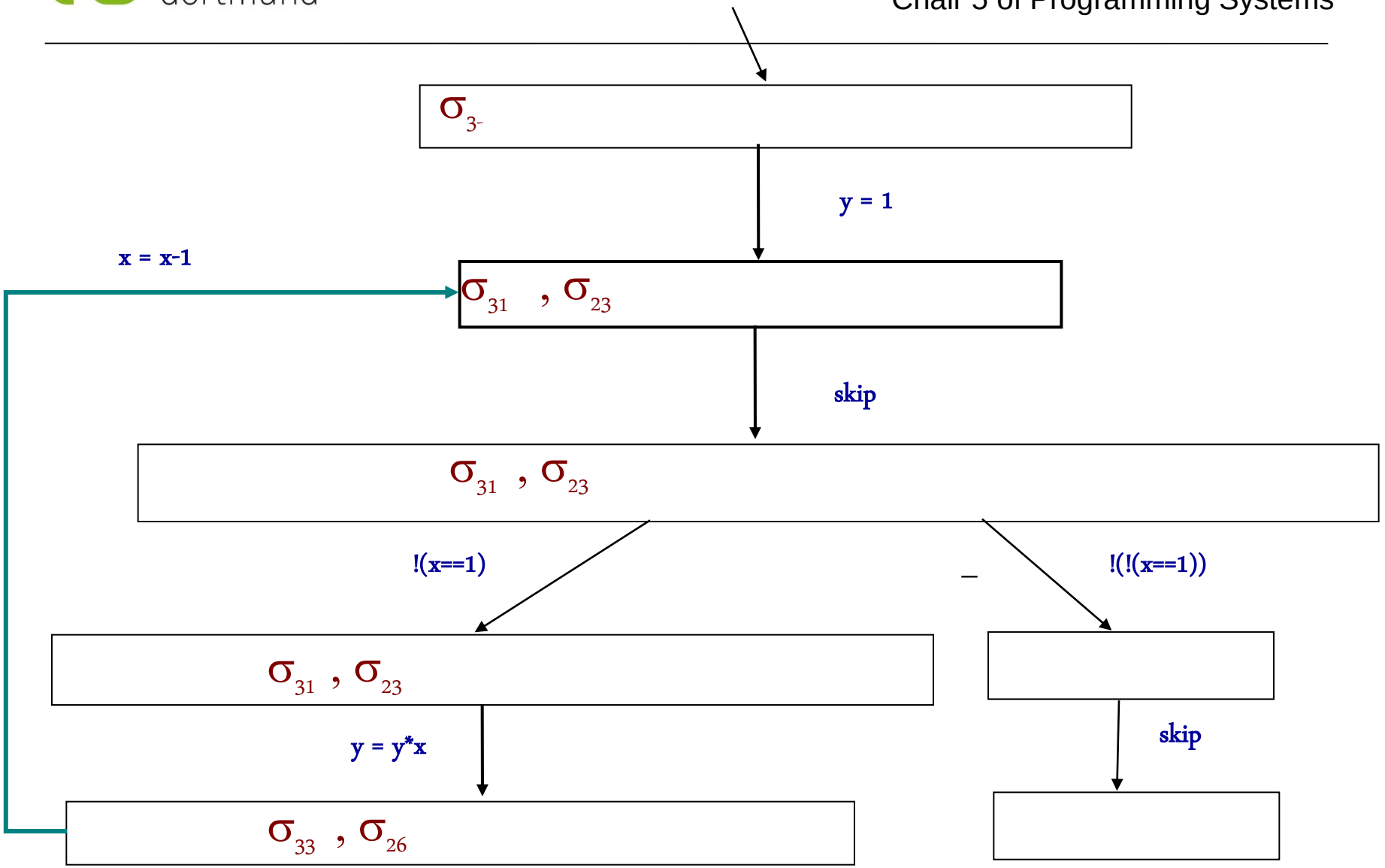
$$[\text{while}_{SOS}]\quad \frac{}{\langle\ \textbf{while}\ (b)\ \{S\},\ \bullet\ \rangle \overset{skip}{\Rightarrow} \langle\ \textbf{if}\ (b)\ \{S;\ \textbf{while}\ (b)\ \{S\}\}\ \textbf{else}\ \{skip\},\ \bullet\ \rangle}$$

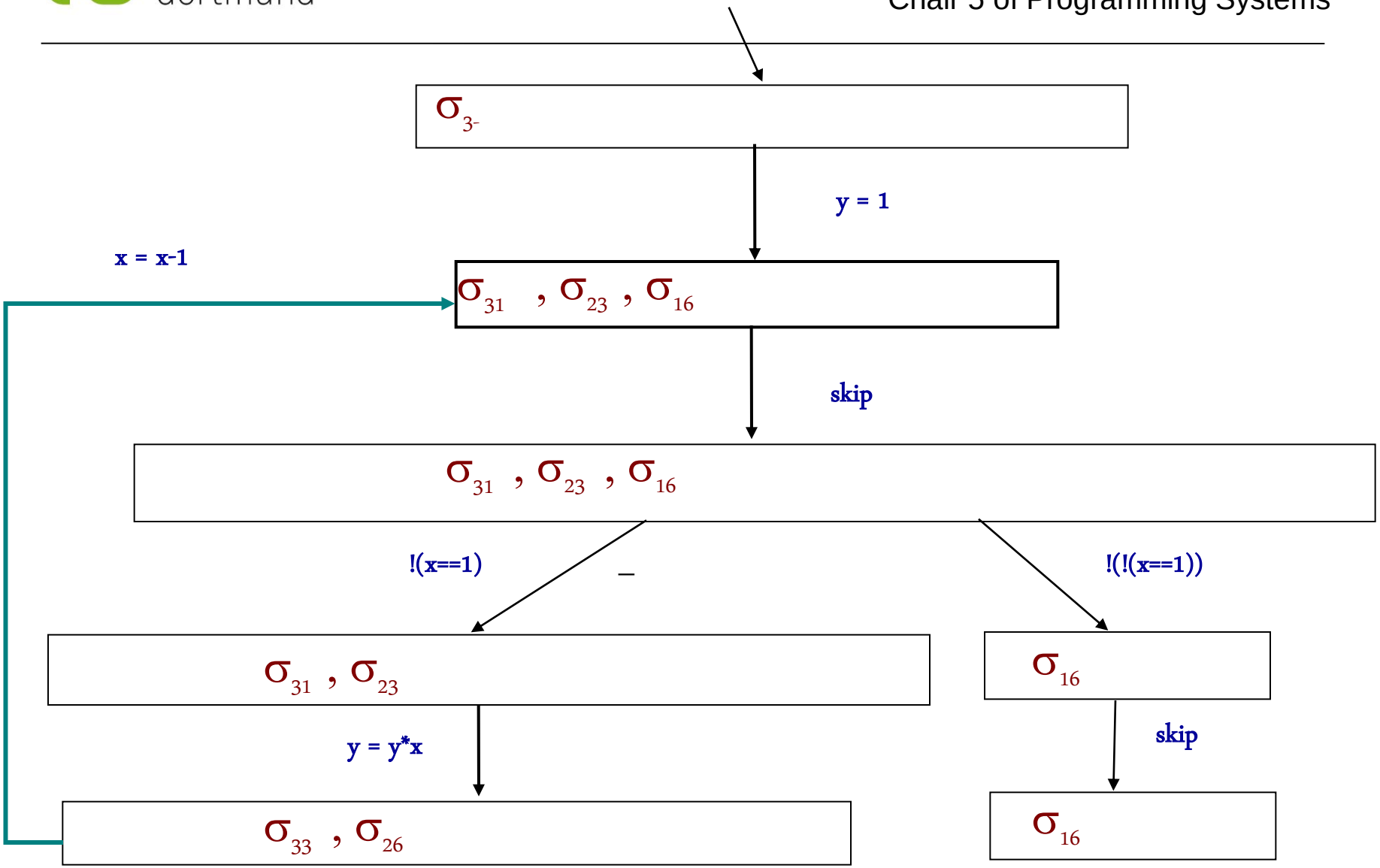

y = 1; while (!(x==1)) {y = z*x; x = x-1}
y = 1
x = x-1
while (!(x==1)) {y = y*x; x = x-1}
skip
if ((!(x==1) {y = z*x; x = x-1; while (!(x==1)) {y = y*x; x = x-1}} else {skip}
!(x==1)
!(!(x==1))
y = z*x; x = x-1; while (!(x==1)) {y = y*x; x = x-1}}
skip
y = y*x
skip
x = x-1; while (!(x==1)) {y = y*x; x = x-1}}

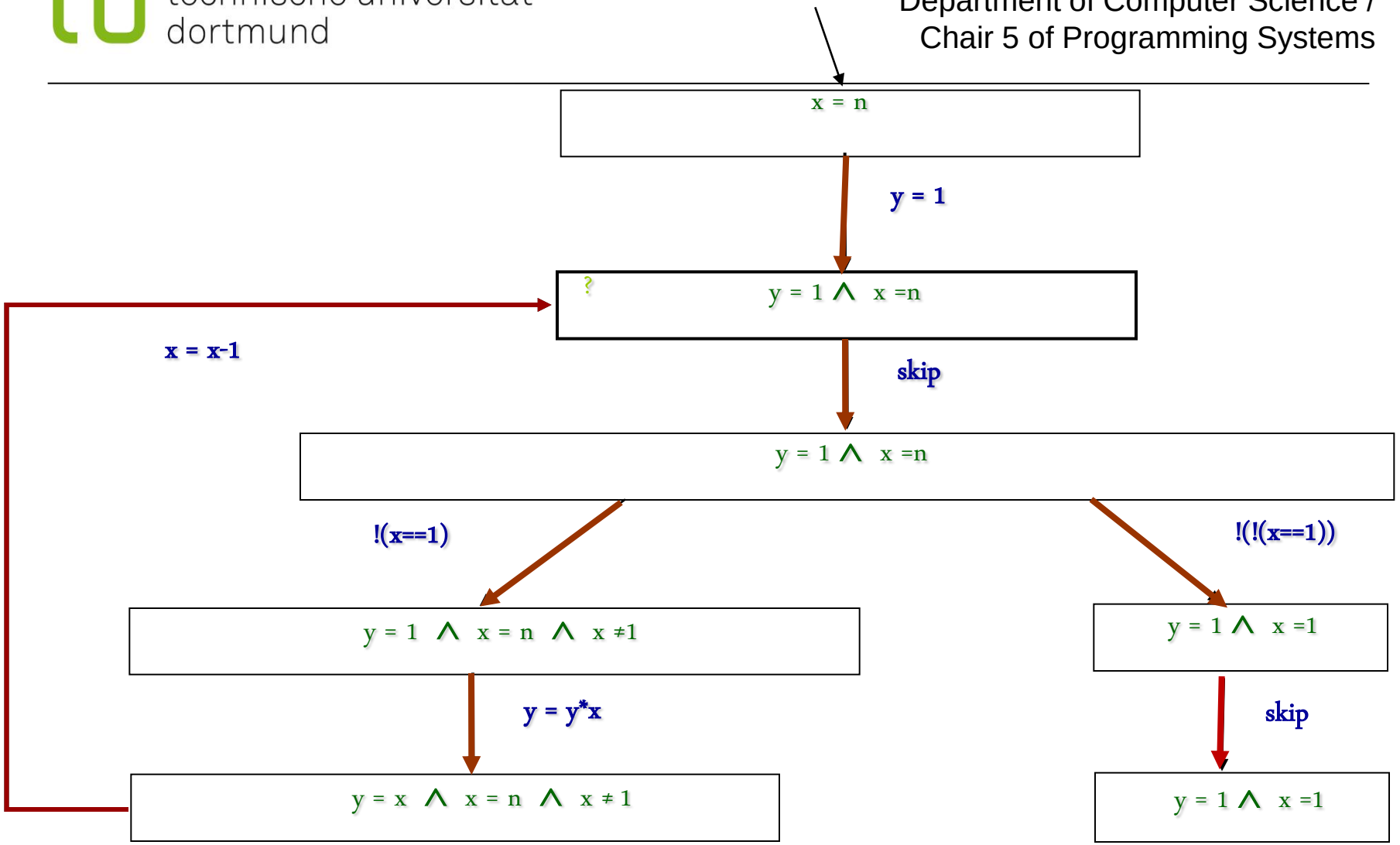
# Label Transition System

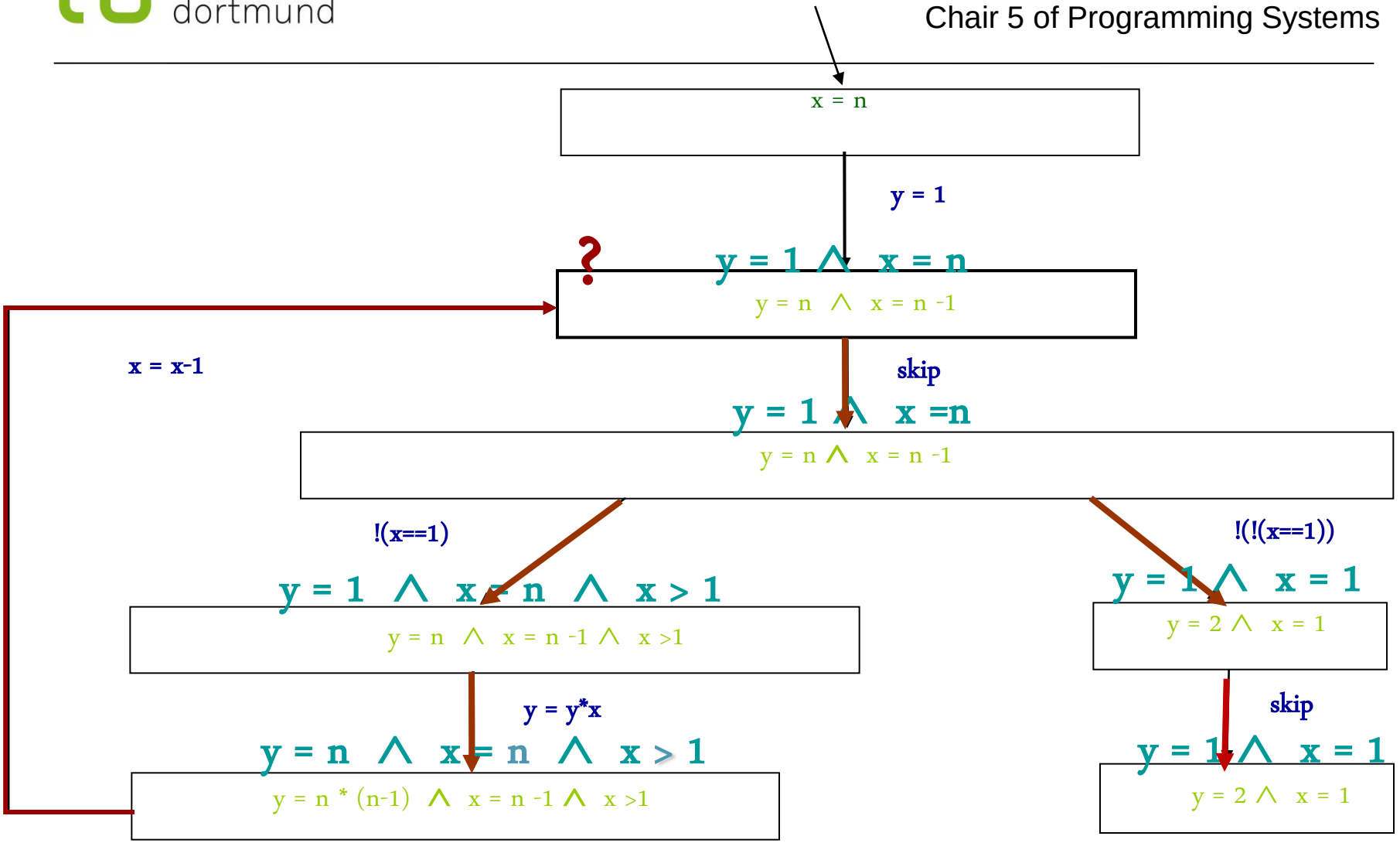


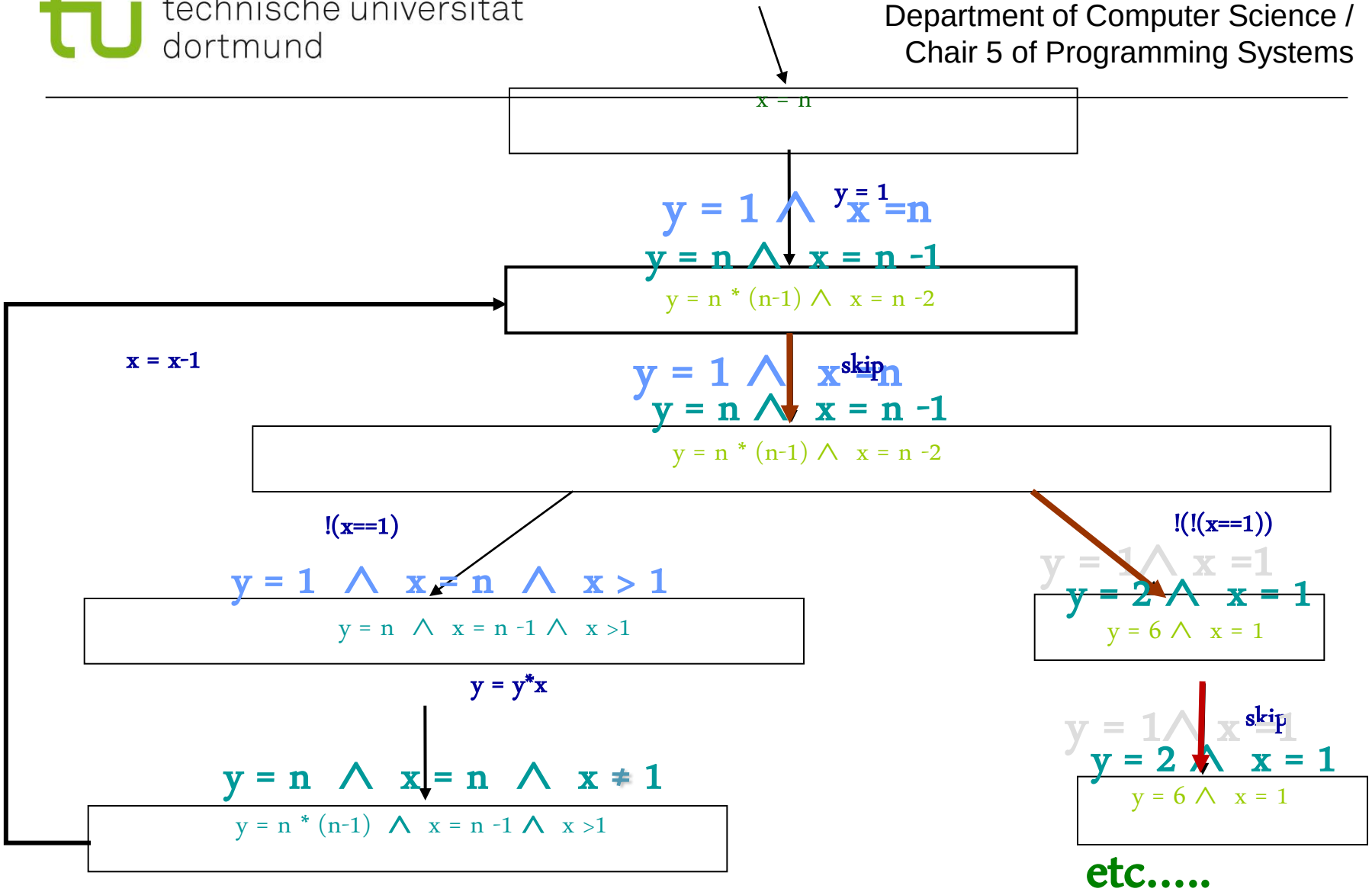

σ3-
y = 1
x = x-1
σ31
skip
σ31
!(x==1)
!(!(x==1))
σ31
y = y*x
skip
σ33


σ3-
y = 1
x = x-1
σ31 , σ23
skip
σ31 , σ23
!(x==1)
!(!(x==1))
σ31 , σ23
skip
y = y*x
σ33 , σ26


σ3-
y = 1
x = x-1
σ31 , σ23 , σ16
skip
σ31 , σ23 , σ16
!(x==1)
!(!(x==1))
σ31 , σ23
σ16
y = y*x
skip
σ33 , σ26
σ16


x = n
y = 1
?
y = 1 ∧ x =n
x = x-1
skip
y = 1 ∧ x =n
!(x==1)
!(!(x==1))
y = 1 ∧ x = n ∧ x ≠1
y = 1 ∧ x =1
y = y*x
skip
y = x ∧ x = n ∧ x ≠ 1
y = 1 ∧ x =1

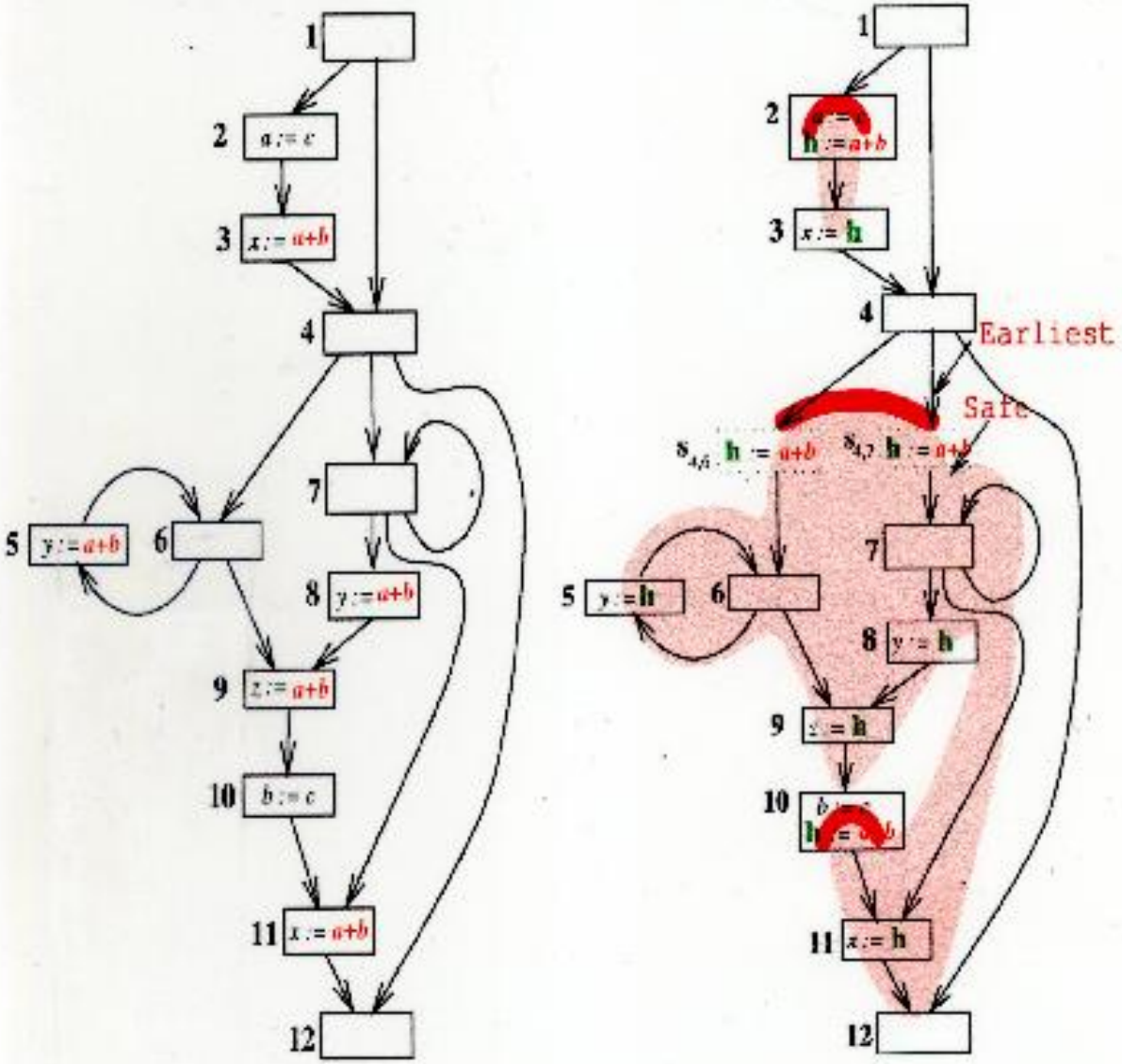

x = n
y = 1
?
y = 1 ∧ x = n
y = n ∧ x = n -1
x = x-1
skip
y = 1 ∧ x =n
y = n ∧ x = n -1
!(x==1)
!(!(x==1))
y = 1 ∧ x = n ∧ x > 1
y = n ∧ x = n -1 ∧ x >1
y = 1 ∧ x = 1
y = 2 ∧ x = 1
y = y*x
skip
y = n ∧ x = n ∧ x > 1
y = n * (n-1) ∧ x = n -1 ∧ x >1
y = 1 ∧ x = 1
y = 2 ∧ x = 1


x = n
y = 1 ∧ x =n
y = n ∧ x = n -1
y = n * (n-1) ∧ x = n -2
x = x-1
y = 1 ∧ x =n
y = n ∧ x = n -1
skip
y = n * (n-1) ∧ x = n -2
!(x==1)
!(!(x==1))
y = 1 ∧ x = n ∧ x > 1
y = n ∧ x = n -1 ∧ x >1
y = 2 ∧ x = 1
y = 6 ∧ x = 1
y = y*x
y = n ∧ x = n ∧ x ≠ 1
y = n * (n-1) ∧ x = n -1 ∧ x >1
skip
y = 2 ∧ x = 1
y = 6 ∧ x = 1
etc.....

# Outline

- Motivation
- Control Flow
- **Static Analysis as Model Checking**
- Conclusions

# Busy Expression Motion




Pre **S** Post

**Is modelled as**

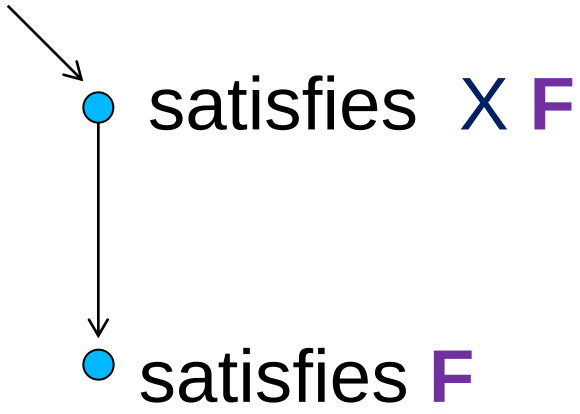
Pre → **S** → Post

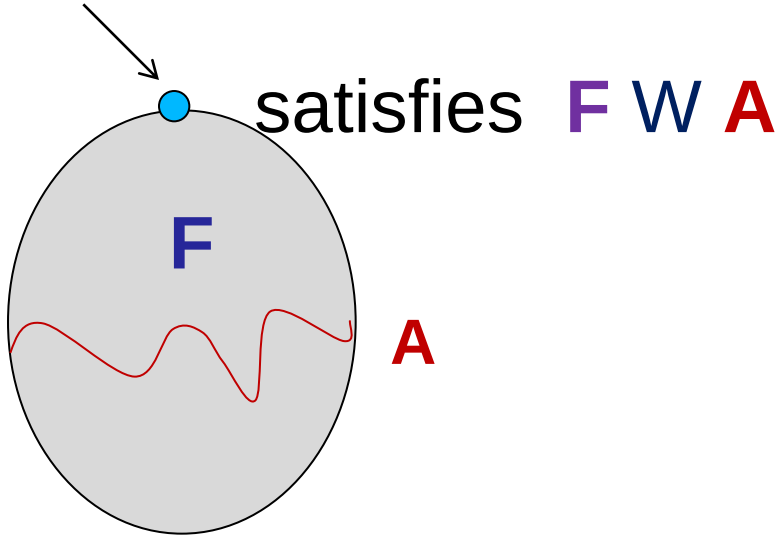
**Easily obtained with SOS via**

Pre —**S**→ Post

# The (Linear Time) Logic

F::= AP | not F | F or F | X F | bX F | F W F | F bW F




In the Graph Model View

# Busy Code Motion

- **Downsafe**

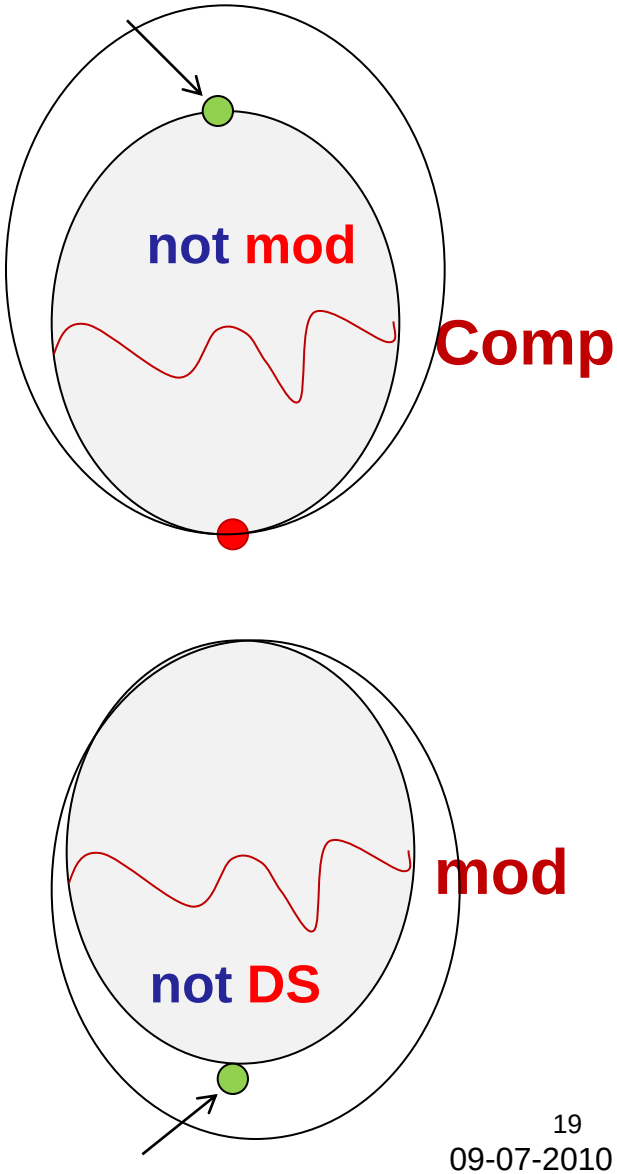
  (not (End or <mod>tt)) **W** comp

- **Earliest**

  bX ( (not **DS**) **bW** mod )

**The computations points**

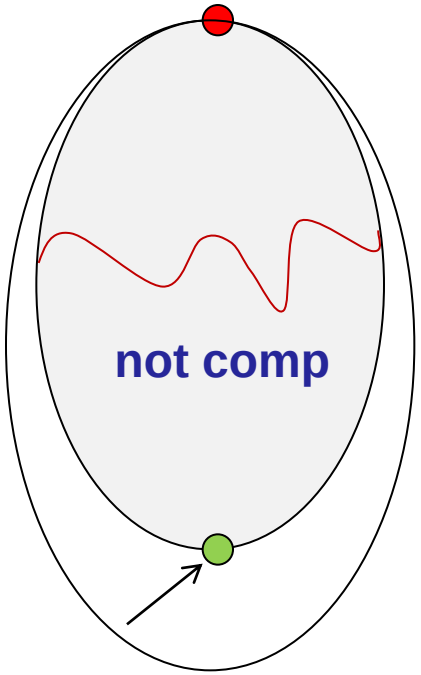
**Downsafe and Earliest!**



# Lazy Code Motion

## 1. Delayed

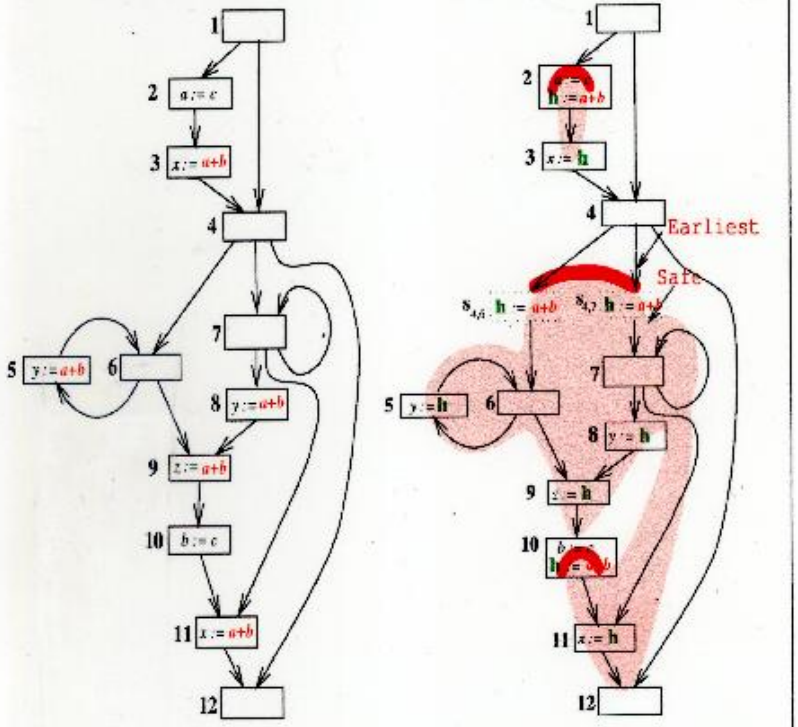
(not (start or **comp**) **bW** Earliest




## The computation points)

**Delayed and ( X(not Delayed) or comp )**

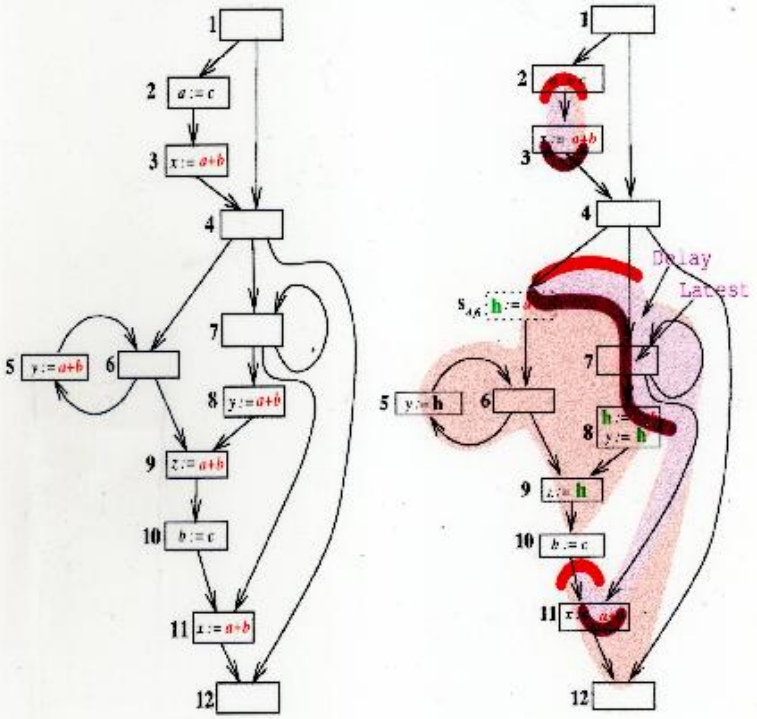
## Busy Expression Motion




**Busy Code Motion is**

**Computationally Optimal**

## Lazy Expression Motion




**Lazy Code Motion is**

**Computationally and Lifetime Optimal**

# Busy Code Motion: Morel Renvoise-Style

## Local Predicates

- $\mathbf{COMP}_\iota(t)$: $\iota$ computes $t$.
- $\mathbf{TRANSP}_\iota(t)$: $\iota$ does not modify any operand of $t$.

## Up-Safety

$$\text{N-USAFE}_\iota = \begin{cases} \mathtt{ff} & \text{if } \iota = s \\ \prod_{\widehat{\iota} \in pred(\iota)} \text{X-USAFE}_{\widehat{\iota}} & \text{otherwise} \end{cases}$$

$$\text{X-USAFE}_\iota = (\text{N-USAFE}_\iota + \mathbf{COMP}_\iota) \cdot \mathbf{TRANSP}_\iota$$

## Down-Safety

$$\text{N-DSAFE}_\iota = \mathbf{COMP}_\iota + \text{X-DSAFE}_\iota \cdot \mathbf{TRANSP}_\iota$$

$$\text{X-DSAFE}_\iota = \begin{cases} \texttt{ff} & \text{if } \iota = e \\ \prod_{\hat{\iota} \in succ(\iota)} \text{N-DSAFE}_{\hat{\iota}} & \text{otherwise} \end{cases}$$

## Insertion Points („Earliestness“)

$$\text{N-INSERT}^{\text{BCM}}_\iota =_{df} \text{N-DSAFE}^\star_\iota \cdot \prod_{\hat{\iota} \in pred(\iota)} (\overline{\text{X-USAFE}^\star_{\hat{\iota}} + \text{X-DSAFE}^\star_{\hat{\iota}}})$$

$$\text{X-INSERT}^{\text{BCM}}_\iota =_{df} \text{X-DSAFE}^\star_\iota \cdot \overline{\mathbf{TRANSP}_\iota}$$

# Morel Renvoise Classical Formulation

## Local Predicates

- $\mathbf{COMP}_\iota(t)$: $\iota$ computes $t$.
- $\mathbf{TRANSP}_\iota(t)$: $\iota$ does not modify any operand of $t$.

## Availability

$$\mathbf{AVIN}(n) = \begin{cases} \mathtt{ff} & \text{if } n = \mathbf{s} \\ \prod_{m \in pred(n)} \mathbf{AVOUT}(m) & \text{otherwise} \end{cases}$$

$$\mathbf{AVOUT}(n) = \mathbf{TRANSP}(n) * (\mathbf{COMP}(n) + \mathbf{AVIN}(n))$$

## Anticipability

$$\mathbf{ANTIN}(n) = \mathbf{COMP}(n) + \mathbf{TRANSP}(n) * \mathbf{ANTOUT}(n)$$

$$\mathbf{ANTOUT}(n) = \begin{cases} \mathtt{ff} & \text{if } n = \mathrm{e} \\ \prod_{m \in succ(n)} \mathbf{ANTIN}(m) & \text{otherwise} \end{cases}$$

## Partial Availability

$$\mathbf{PAVIN}(n) = \begin{cases} \mathtt{ff} & \text{if } n = \mathrm{s} \\ \sum_{m \in pred(n)} \mathbf{PAVOUT}(m) & \text{otherwise} \end{cases}$$

$$\mathbf{PAVOUT}(n) = \mathbf{TRANSP}(n) * (\mathbf{COMP}(n) + \mathbf{PAVIN}(n))$$

## Placement Possible

$$\mathbf{PPIN}(n) = \begin{cases} \mathtt{ff} & \text{if } n = \mathrm{s} \\ \mathbf{CONST}(n) * \\ (\prod_{m \in pred(n)} (\mathbf{PPOUT}(m) + \mathbf{AVOUT}(m)) * \\ (\mathbf{COMP}(n) + \mathbf{TRANSP}(n) * \mathbf{PPOUT}(n)) & \text{otherwise} \end{cases}$$

$$\mathbf{PPOUT}(n) = \begin{cases} \mathtt{ff} & \text{if } n = \mathrm{e} \\ \prod_{m \in succ(n)} \mathbf{PPIN}(m) & \text{otherwise} \end{cases}$$

with $\mathbf{CONST}(n) =_{df} \mathbf{ANTIN}(n) * (\mathbf{PAVIN}(n) + \neg\mathbf{COMP}(n) * \mathbf{TRANSP}(n))$

# Truly Bi-Directional

# Initialization

$$\mathbf{INSIN}(n) =_{df} \texttt{ff}$$

$$\mathbf{INSOUT}(n) =_{df} \mathbf{PPOUT}(n) * \neg\mathbf{AVOUT}(n) * (\neg\mathbf{PPIN}(n) + \neg\mathbf{TRANSP}(n))$$

$$\mathbf{REPLACE}(n) =_{df} \mathbf{COMP}(n) * \mathbf{PPIN}(n)$$

# Conceptual Difference

**Besides the improved Model Structure:**

- **Two** hierachical
- greatest,
- uni-directional fixpoint computations

- **Four** hierachical
- greatest and **least**,
- **bi-**directional fixpoint computations

**Impact e.g.:**

- Correctness and Optimality Proof
- Refinement (Lazy Code Motion)

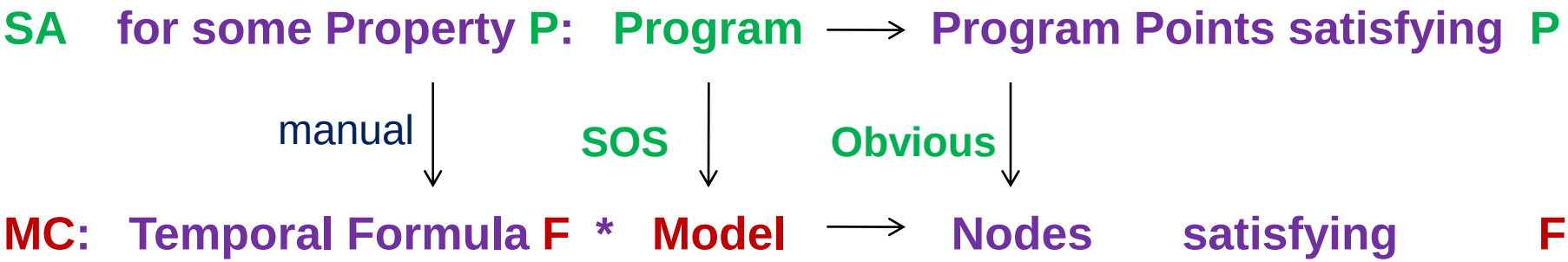
# Outline

- Motivation
- Control Flow
- Static Analysis as Model Checking
- **Conclusions**

# Conclusions

- The SA-Generator
- Models and Logics
- Modular Proofs
- Property-Oriented Expansion

# SA Generation



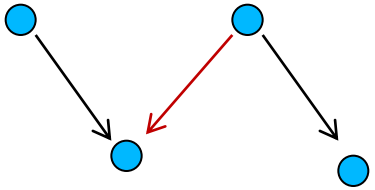

**Generated analyses were more efficient than the handwritten counterparts!**

- Better structure
- Optimized Model Checkers (Fixpoint Analysis Machine)
- Dwyer and Robby example

# Models and Logics

## Flow Graph satisfies **Property**

Critical Edges



## (bi-partite) Kripke Structure satisfies F W not**A**

Pre **S** Post becomes Pre → **S** → Post

## Kripke Transitions System satisfies $AG_A F$

Pre **S** Post becomes Pre —**S**→ Post

# **Modular Proofs** at the What-Level

A step in the optimality proof
(for bi-partite Kripke Structures)

Safe = DS and US,
Earliest implies (comp or not DS)

delivers

Safe and Earliest **=** DS and Earliest

# Property-Oriented Expansion

| Classical | | Essence – oriented |
|---|---|---|
| **Syntactic**<br>parse tree | **Specialized**<br>flow graphs | **Semantic**<br>abstractly interpreted transition systems |
| **Computation oriented**<br>attribute evaluation | equational systems | **Property- oriented**<br>modal formulas |
| **Fixed Structure**<br>invariant flow of control | **Specialized Structures**<br>no critical edges placement models | **Evolving Structures**<br>property – oriented expansion |

# Conclusion

**HOW**

Static Analysis

**WHAT**

Model Checking

„Design for“ Paradigms

Towards domain-specific **simple** solutions