

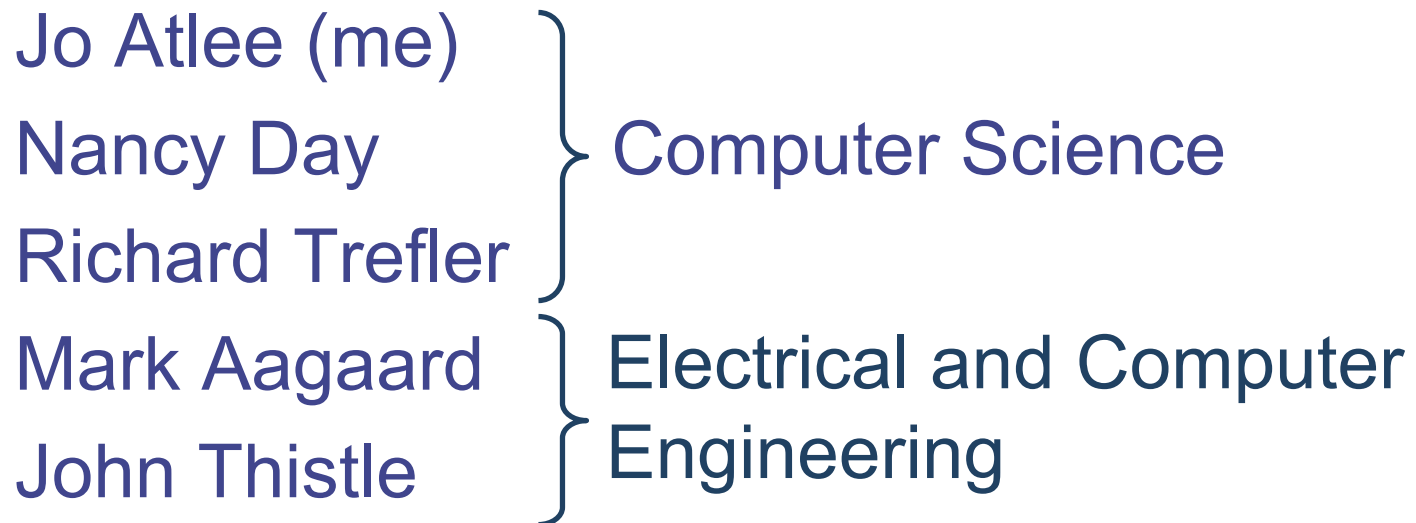
Template Semantics for Model-Based Notations

Jianwei Niu Joanne Atlee Nancy Day

WATFORM
University of Waterloo

Waterloo Formal Methods (WatForm)

The use of mathematics to model and reason about computer systems - usually for the purpose of ensuring that the system will be acceptable.



WatForm Research Programs

- Practical Formalisms: Mathematical languages that are usable by practitioners.
- Automated techniques for analyzing formal models
- Combining analysis techniques to verify larger or new types of problems
- Methodologies for constructing models to facilitate analysis (e.g., abstraction)
- Case Studies

Notation-specific Analyzers

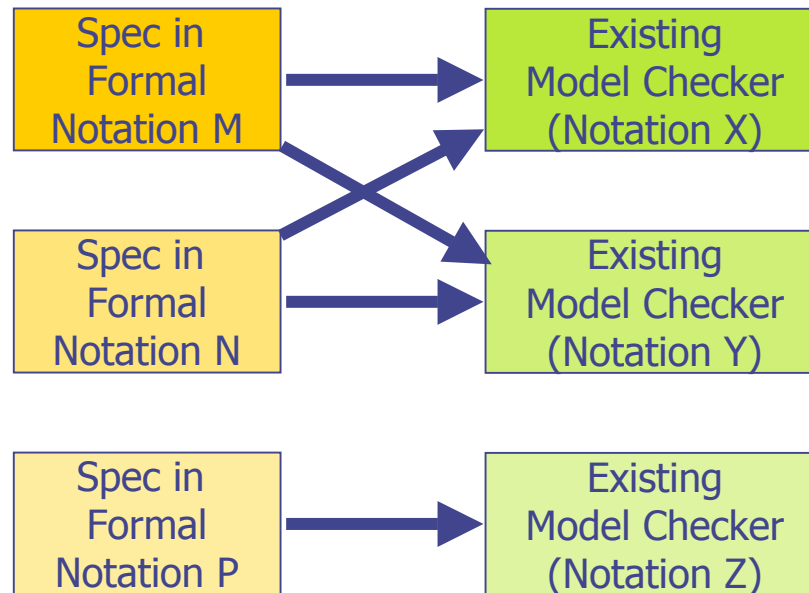
Goal: Want to analyze specifications written in our favourite specification notation

Problems:

- Proprietary or custom specification notations
- Semantic gap between specification notation and analyzer's computation model
- Specification notations have evolving or competing semantics

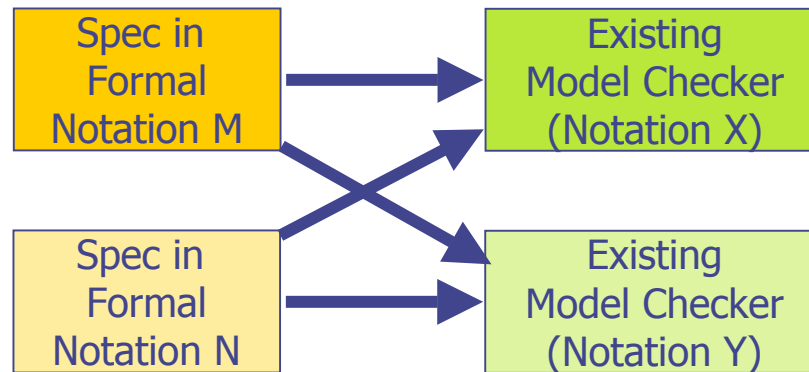
Various Approaches

Direct Translation: [Zave & Jackson, Mikk et al., Chan et al., Sreemani & Atlee, Avrunin & Corbett & Dillon]

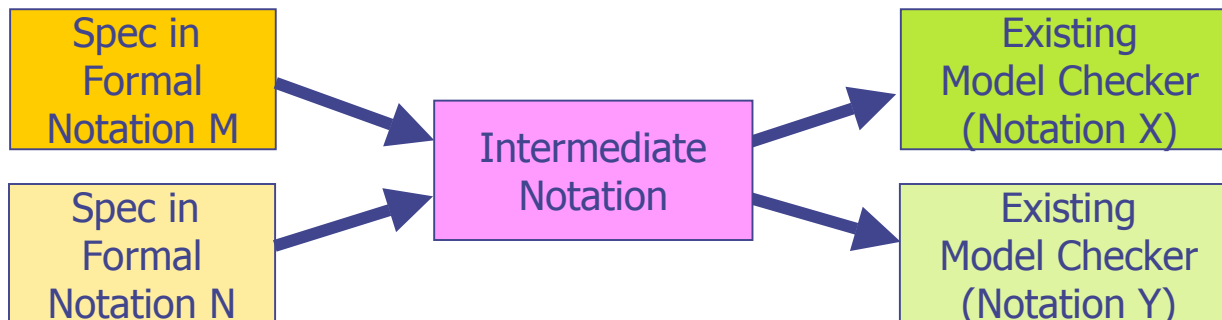


Various Approaches

Direct Translation: [Zave & Jackson, Mikk et al., Chan et al., Sreemani & Atlee, Avrunin & Corbett & Dillon]

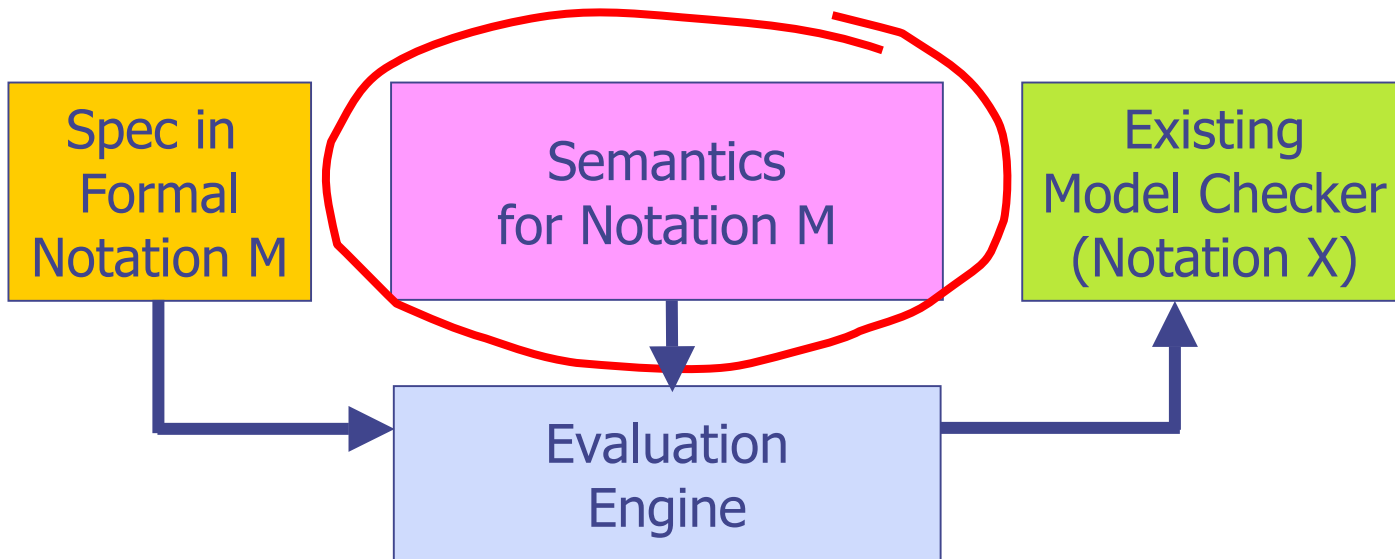


Intermediate Notations: [Bensalem et al., Bosza et al.]



Various Approaches

Generate analyzers from notation's semantics:
[Cleaveland & Sims, Dillon & Stirewalt, Pezzè & Young, Day & Joyce]

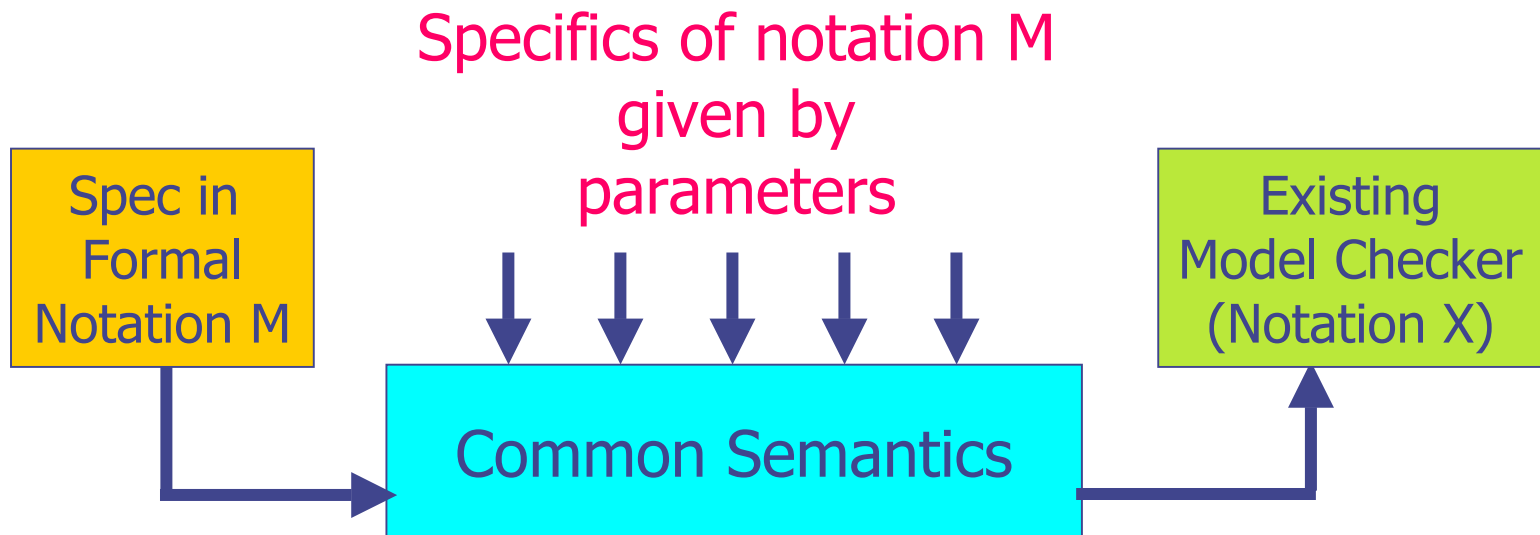


- Hard to write
- Incomplete (no data variables)

Template Semantics

We propose a template-based approach to defining model-based notations:

- Semantics common to notations are pre-defined as parameterized predicates in the template
- A notation's distinct semantics are specified as parameters



Today's Talk

Template Semantics

- Template definitions
- Template parameters
- Step semantics
- Composition operators

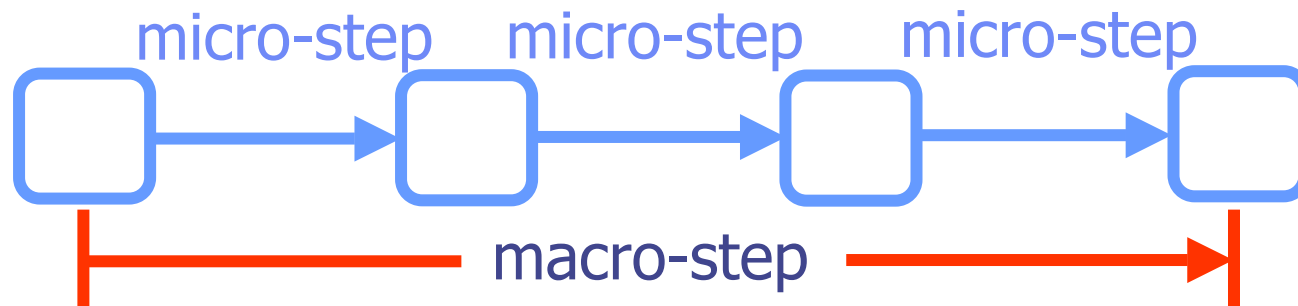
Generating Analyzers from Template Definitions

Computation Model

- **Hierarchical Transition Systems (HTS)**, with:
 - States (hierarchical)
 - Internal events
 - External events
 - Variables
 - Transitions
 - <source, trigger, condition, assignments, gen_events, dest>
- **No Concurrency:** concurrency introduced when composing multiple HTSs

Semantics of HTS

- **Snapshot:** observable point in execution
- **Operational Semantics:** a relation over pairs of consecutive snapshots (**steps**)
 - **micro-steps:** execute a single transition
 - **macro-steps:** execute a sequence of micro-steps until a stable state is reached



Semantics of HTS

- **Snapshot:** observable point in execution

	CS = current states
Basic	IE = current internal events
Elements	AV = current variable values
	O = generated external events

Semantics of HTS

- **Snapshot:** observable point in execution

	CS = current states
Basic	IE = current internal events
Elements	AV = current variable values
	O = generated external events

	CSa	
Auxiliary	IEa	Used to determine which transitions are enabled
Elements	AVa	
	la	

Step Semantics of HTS

Common semantics

- **enabled transitions**: identifies which transitions that are enabled by the enabling states, events, and variable values
- **apply**: apply a transition's effects to snapshot
- **init**: initialize snapshot at start of macro-step
- **micro-step**: selects an enabled transition and applies its actions (destination state, variable-value assignments) to the snapshot
- **macro-step**: sequence of micro-steps, terminating with a stable state (in which no transition is enabled)

Step Semantics of HTS

Common semantics

- enabled transitions
- apply
- init

Template parameters

- enabling states
- enabling events
- enabling variable values
- change state
- generate events
- change variable values
- initialize state info
- initialize event info
- initialize variable info

Template Parameters

how initialized at beginning of macro-step



INIT



NEXT

how changes when a transition is taken

 Enabling States

CS		
IE		
AV		
O		
CSa		
IEa		
AVa		
Ia		
en_states		
en_trig		
en_cond		

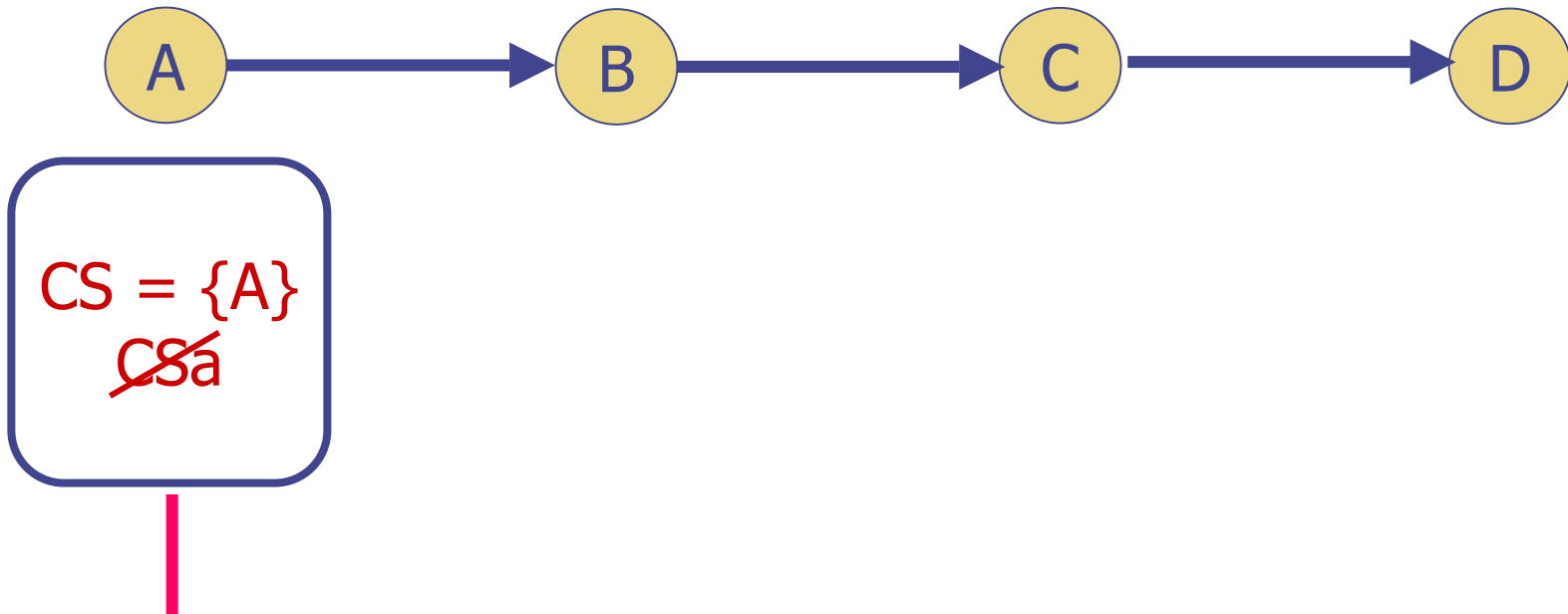


how used to enable a transition

Which states can enable transitions?

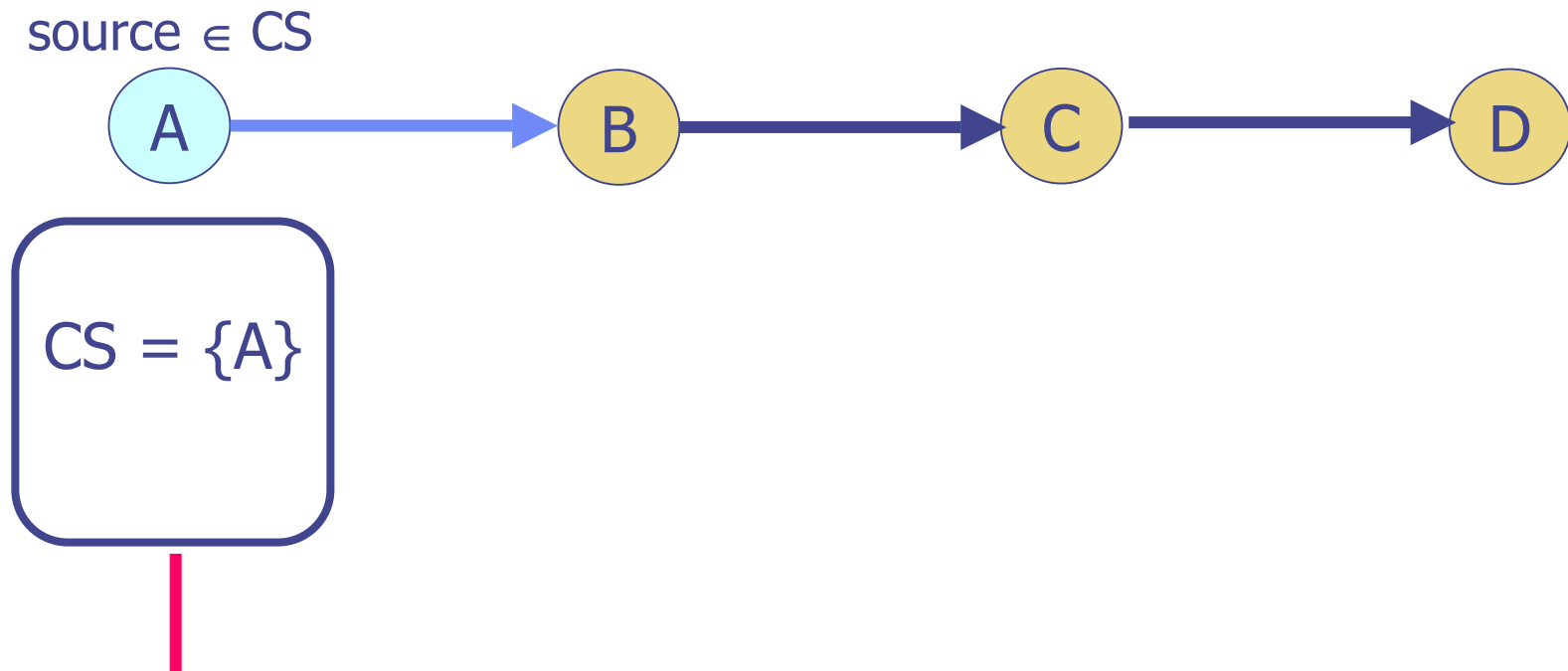
- Options:
- Current states
 - Only current states at the beginning of the macro-step

en_states=
source \in CS



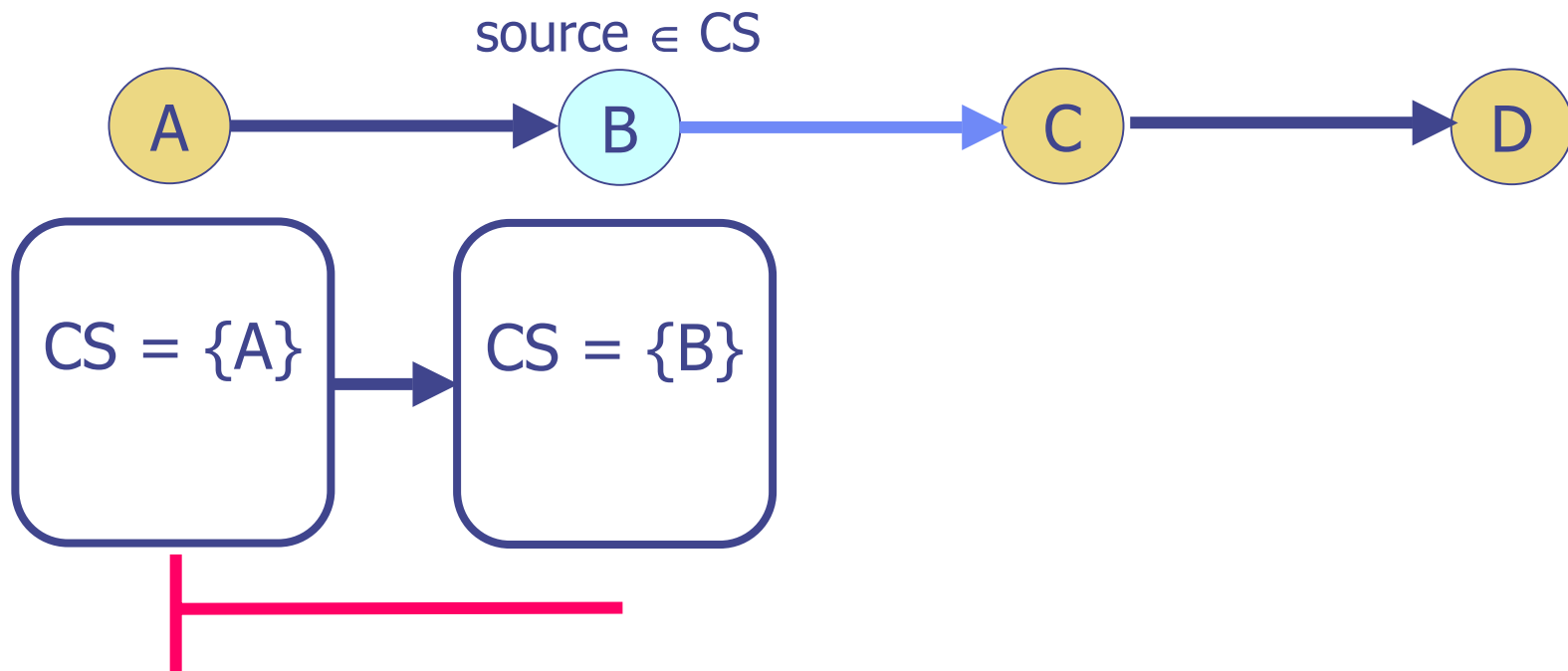
Which states can enable transitions?

- Options:
- Current states
 - Only current states at the beginning of the macro-step



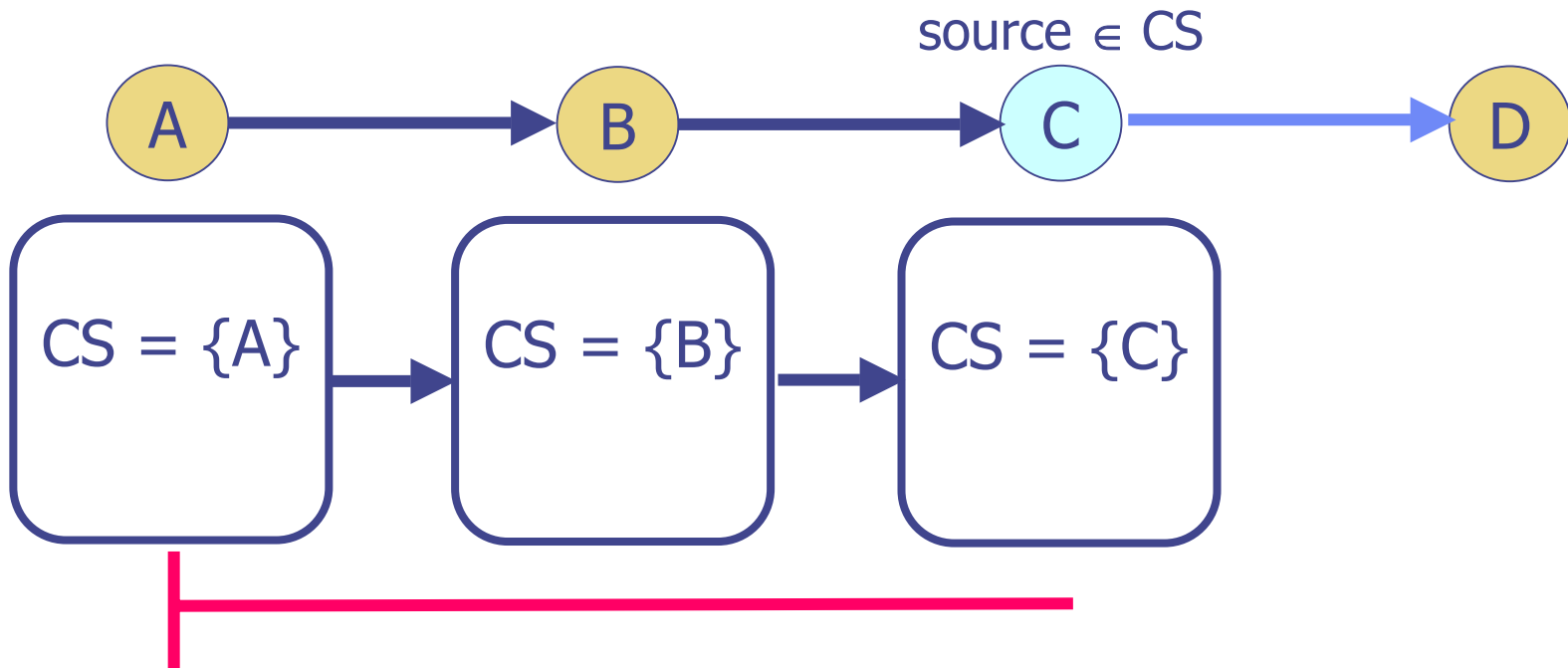
Which states can enable transitions?

- Options:
- Current states
 - Only current states at the beginning of the macro-step



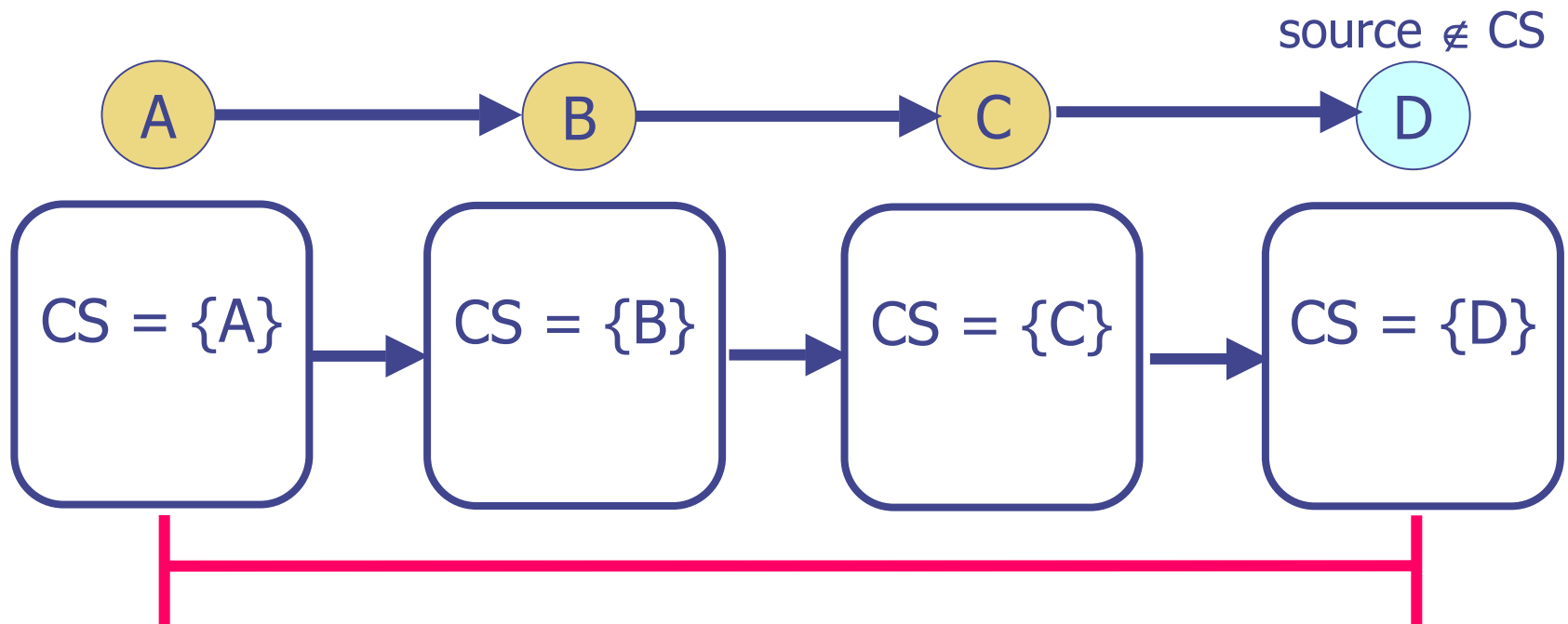
Which states can enable transitions?

- Options:
- Current states
 - Only current states at the beginning of the macro-step



Which states can enable transitions?

- Options:
- Current states
 - Only current states at the beginning of the macro-step

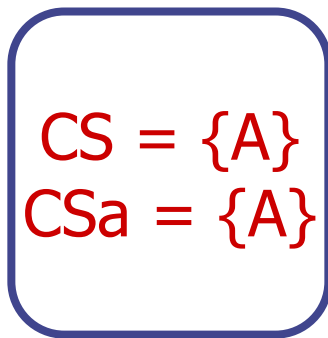


Which states can enable transitions?

- Options:
- Current states
 - Only current states at the beginning of the macro-step

en_states≡

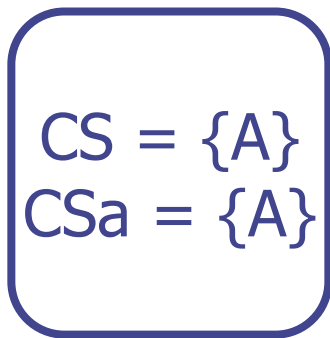
source $\in CS \cap CSa$



Which states can enable transitions?

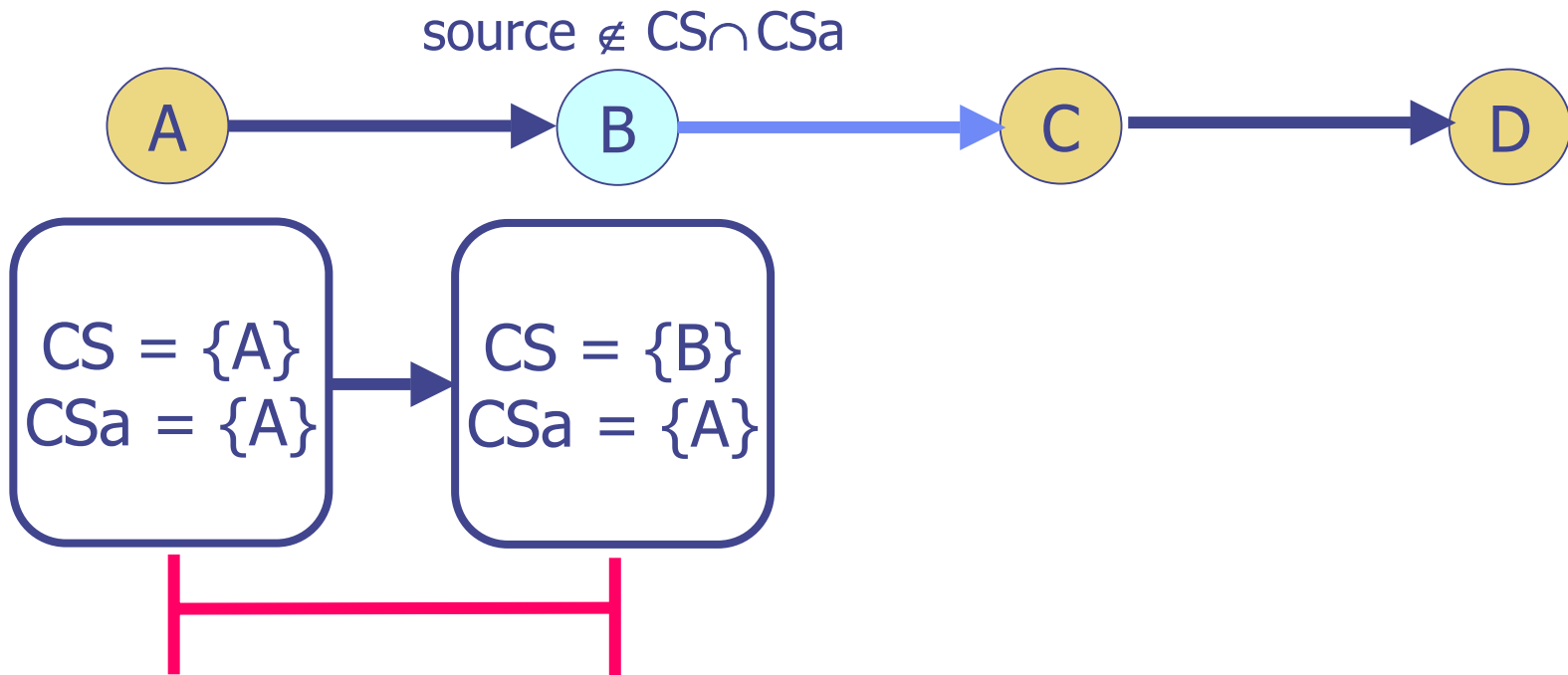
- Options:
- Current states
 - Only current states at the beginning of the macro-step

source $\in CS \cap CSa$



Which states can enable transitions?

- Options:
- Current states
 - Only current states at the beginning of the macro-step



Template Parameters

how initialized at beginning of macro-step



INIT
















NEXT

how changes when a transition is taken

 Enabling States

 Enabling Events

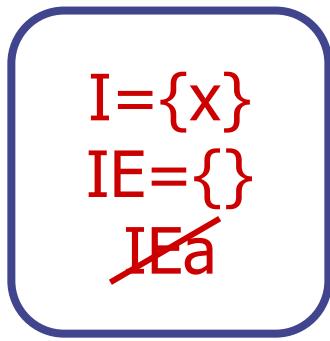
	INIT	NEXT
CS		
IE		
AV		
O		
CSa		
IEa		
AVa		
Ia		
en_states		
en_trig		
en_cond		


how used to enable a transition

Which events trigger transitions?

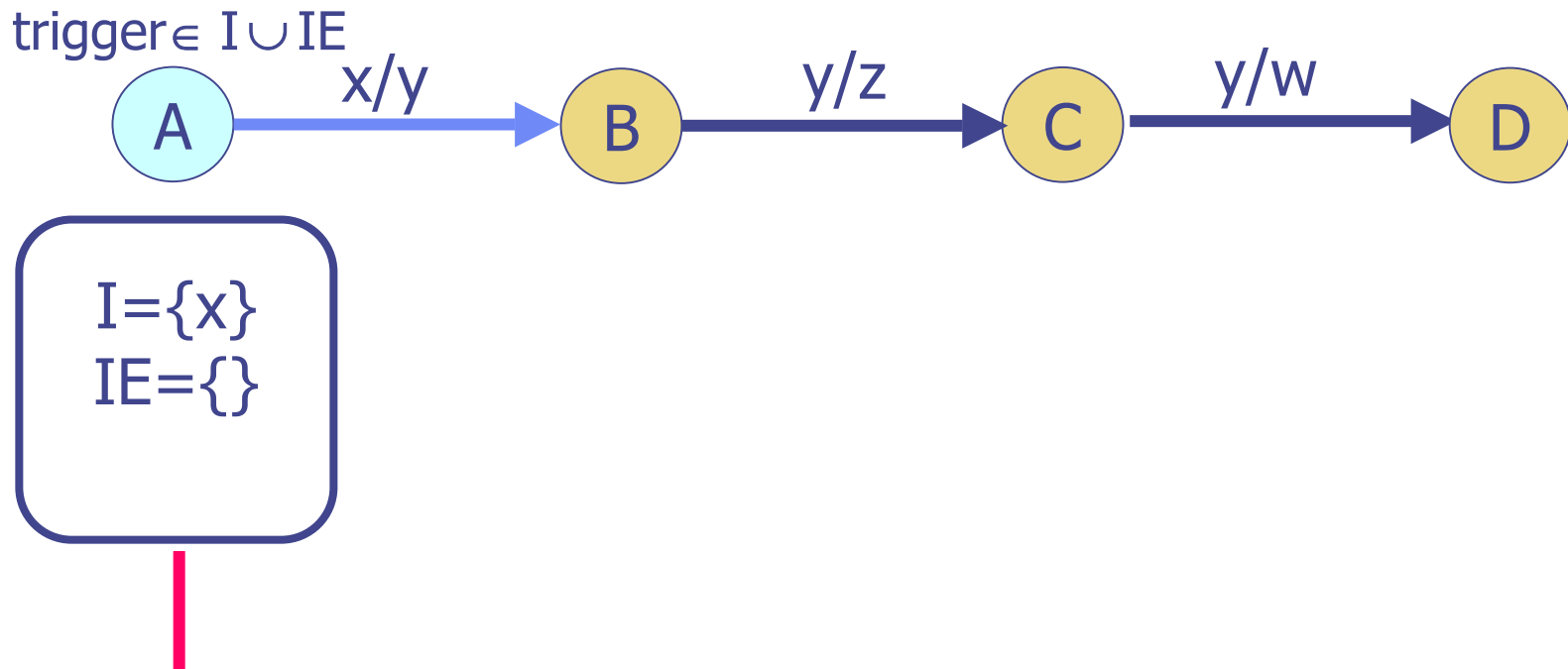
- Options:
- events generated since the beginning of the macro-step
 - events generated in the last micro-step
 - events that haven't been processed

en_trig≡
trigger ∈ I ∪ IE



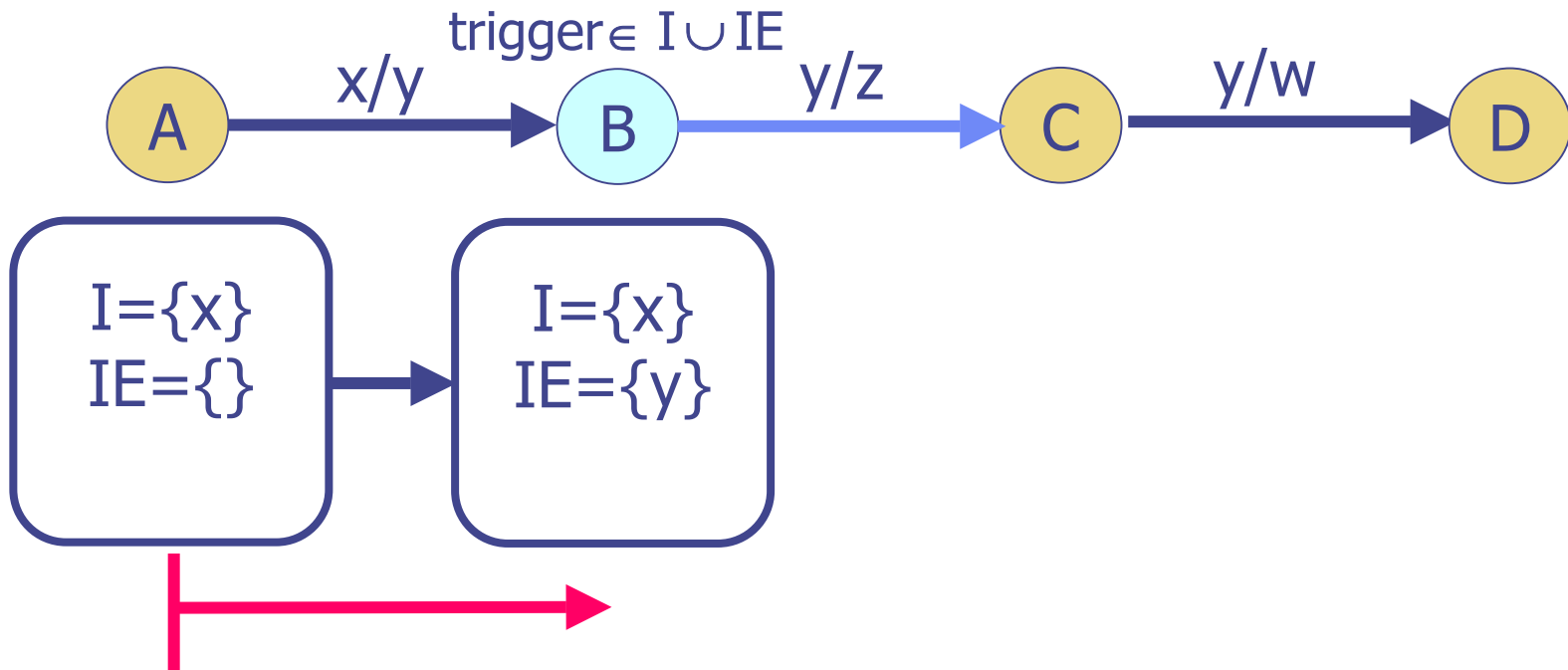
Which events trigger transitions?

- Options:
- events generated since the beginning of the macro-step
 - events generated in the last micro-step
 - events that haven't been processed



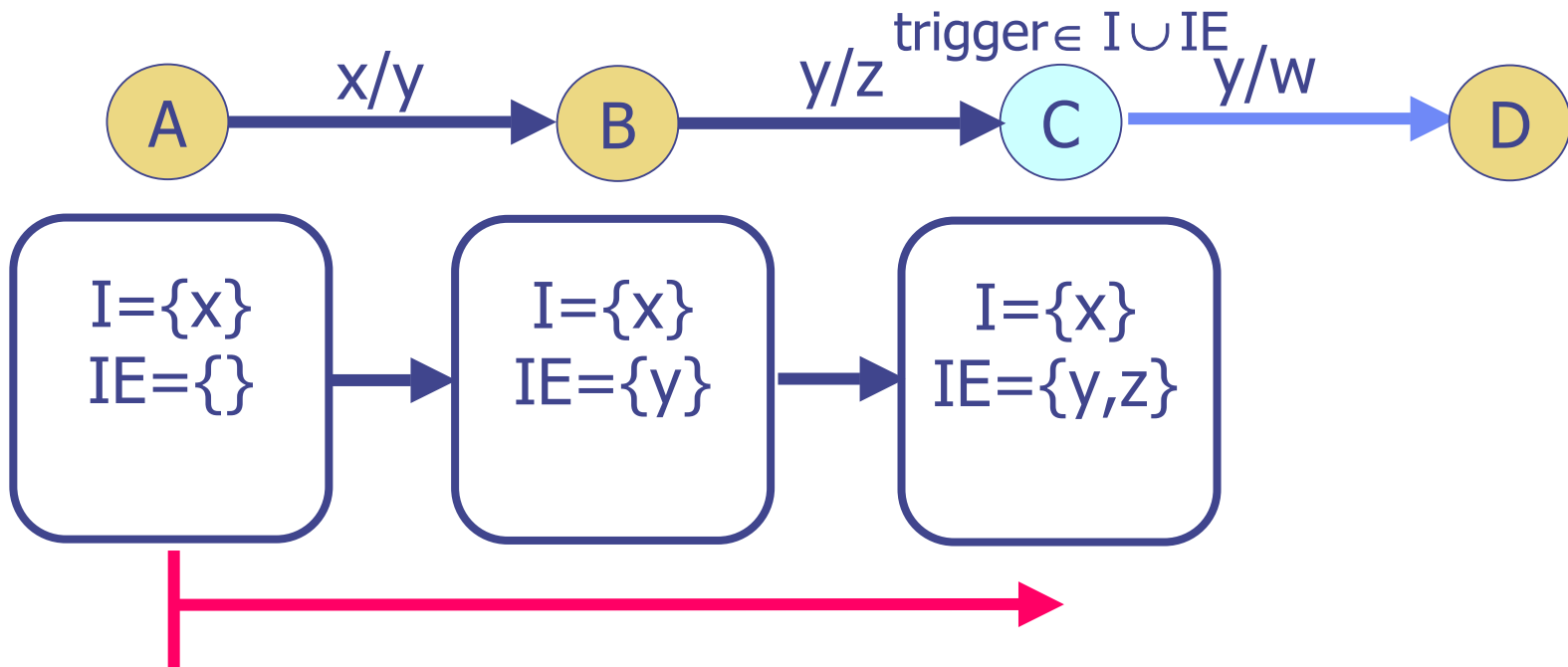
Which events trigger transitions?

- Options:
- events generated since the beginning of the macro-step
 - events generated in the last micro-step
 - events that haven't been processed



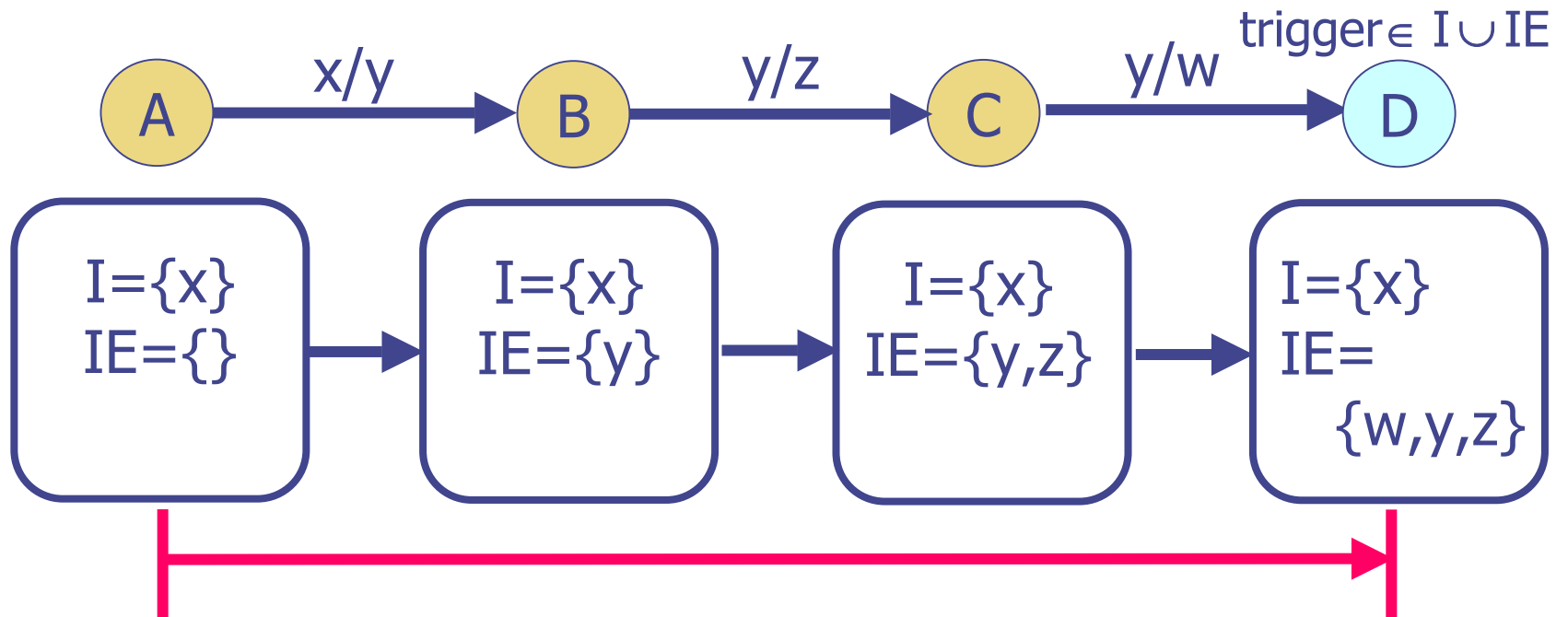
Which events trigger transitions?

- Options:
- events generated since the beginning of the macro-step
 - events generated in the last micro-step
 - events that haven't been processed



Which events trigger transitions?

- Options:
- events generated since the beginning of the macro-step
 - events generated in the last micro-step
 - events that haven't been processed

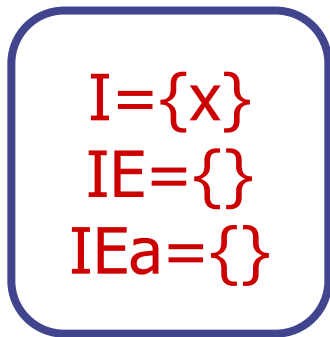


Which events trigger transitions?

- Options:
- events generated since the beginning of the macro-step
 - events generated in the last micro-step
 - **events that haven't been processed**

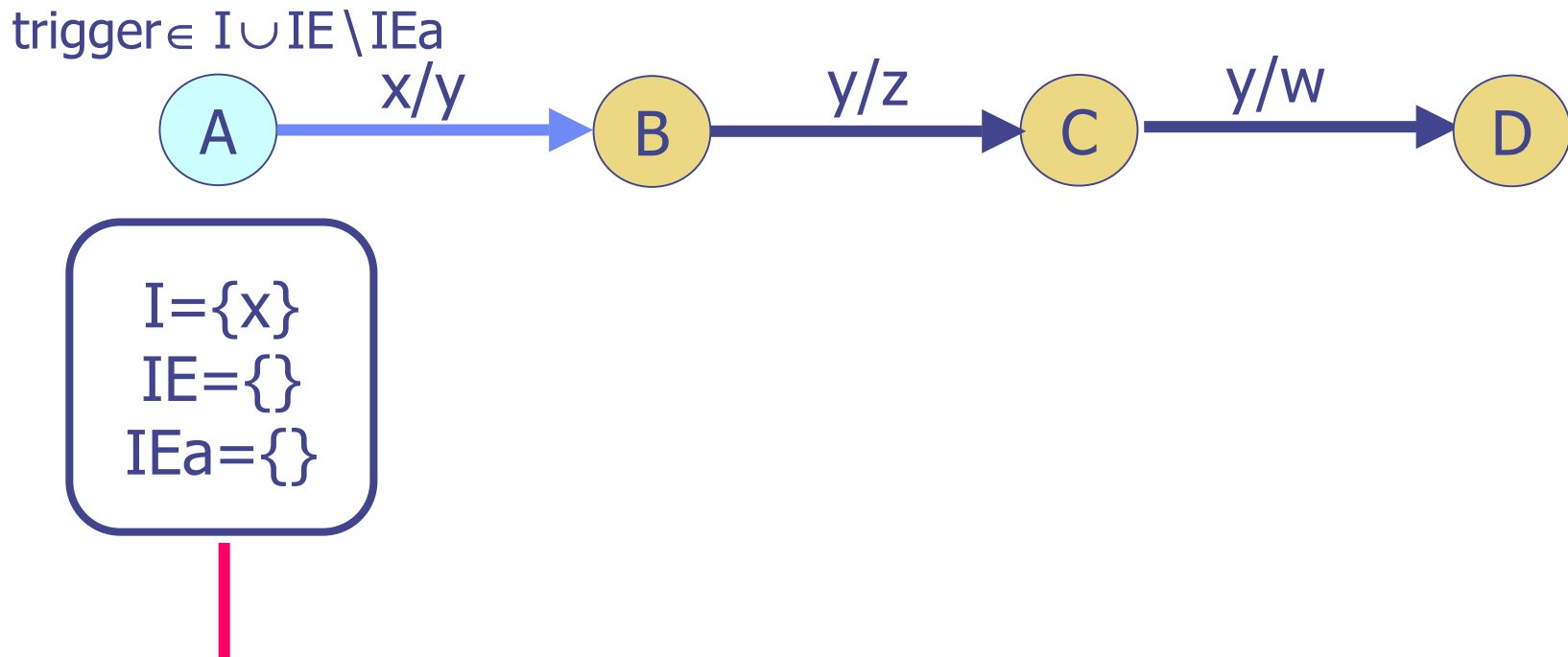
$en_trig \equiv$

$trigger \in I \cup IE \setminus IEa$



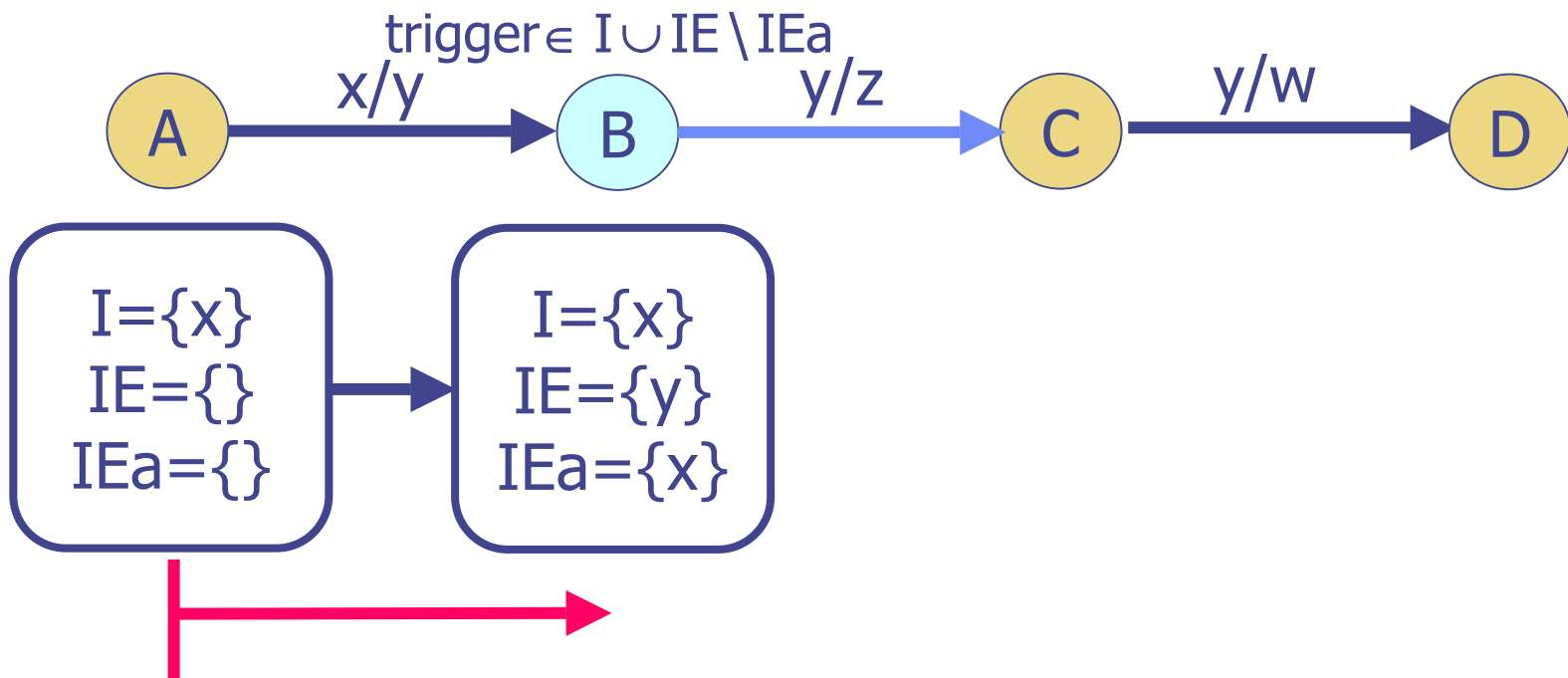
Which events trigger transitions?

- Options:
- events generated since the beginning of the macro-step
 - events generated in the last micro-step
 - **events that haven't been processed**



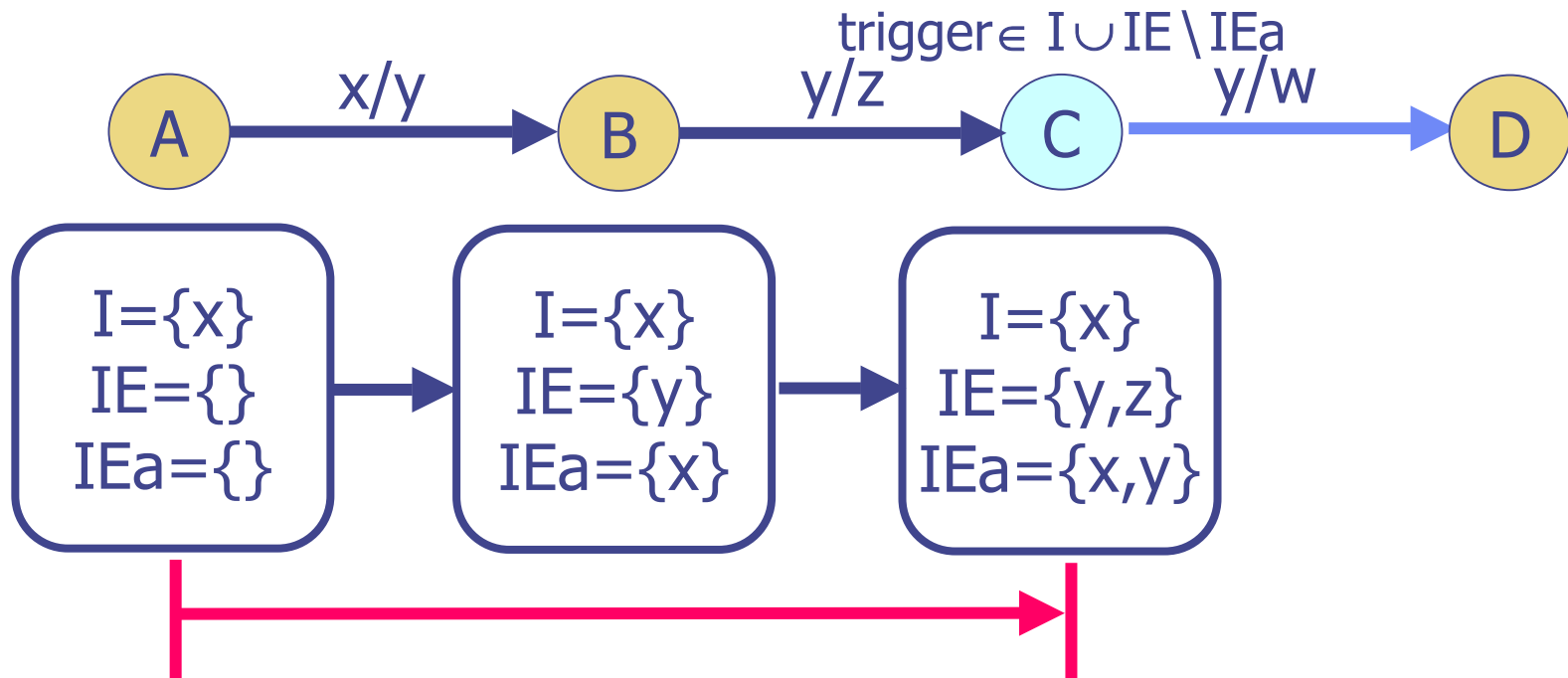
Which events trigger transitions?

- Options:
- events generated since the beginning of the macro-step
 - events generated in the last micro-step
 - **events that haven't been processed**



Which events trigger transitions?

- Options:
- events generated since the beginning of the macro-step
 - events generated in the last micro-step
 - **events that haven't been processed**



Template Parameters

how initialized at beginning of macro-step



INIT

how changes when a transition is taken



NEXT

Enabling States

Enabling Events

Enabling Variables

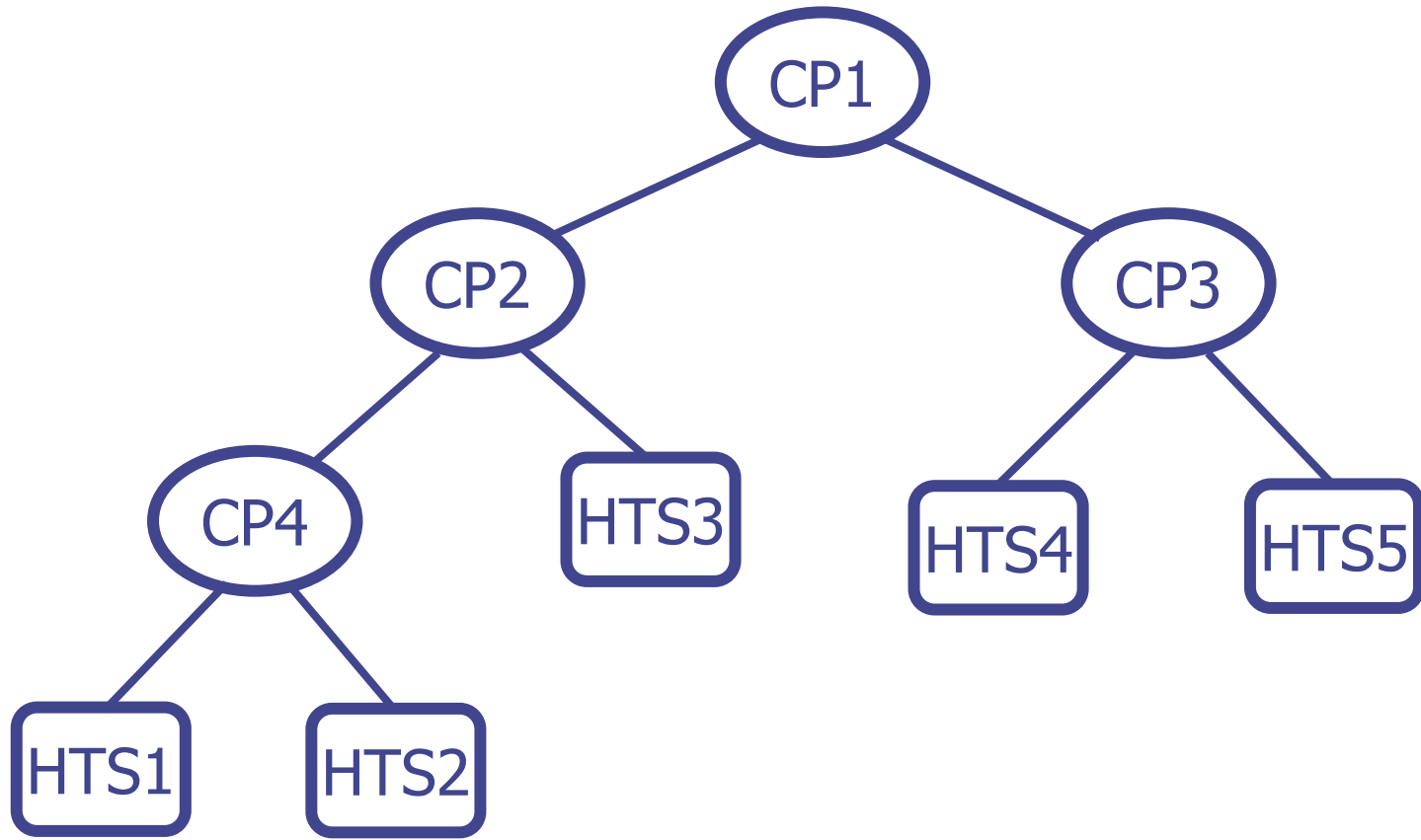
CS	-	CS-source+dest
IE	{ }	IE+gen_events
AV	-	AV \oplus assignments
O	-	-
Csa	CS	CSa
IEa	{ }	IEa+trig
AVa	AV	AVa
Ia	-	-
en_states	source \in CSa	
en_trig	trigger $\in (I \cup IE) \setminus IEa$	
en_cond	AVa \models condition	



how used to enable a transition

Composition Operators

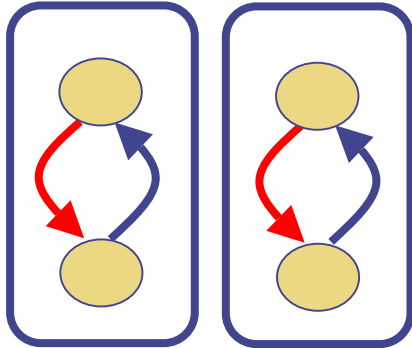
Compose HTSs or collections of HTSs:



Composition Operators

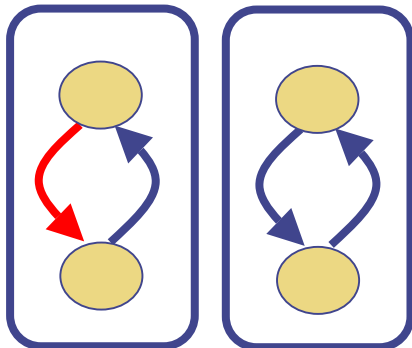
- **Constrain when components can take a step**
- **Share snapshot information:**
 - communication of events
 - consistent values among shared variables

Example: Parallel Composition



Case 1:

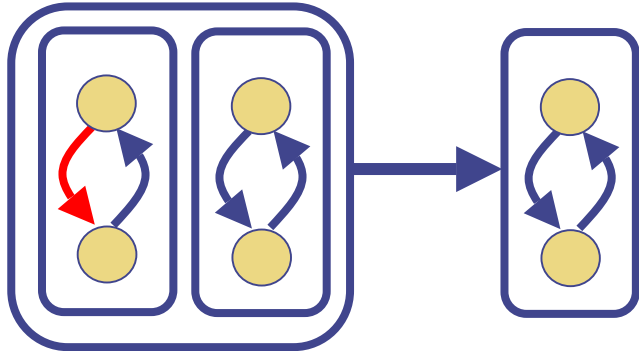
Both components are enabled and execute simultaneously



Case 2:

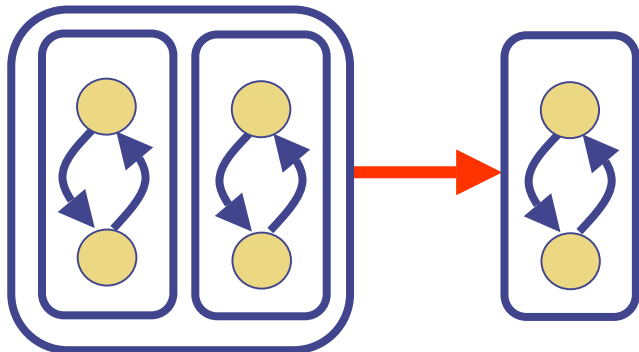
One component is enabled and executes in isolation

Example: Interrupt Composition



Case 1:

One component has control; it executes



Case 2:

Control transfers to from one component to the other

Composition Operators

- Steps are macro-steps as defined by template
- Components share information by updating snapshot elements with data from other components
 - events, shared variables, auxiliary snapshot elements
 - snapshots updated as defined by template-parameters

Interrupt Composition

$$\begin{aligned}
 & N_{micro}^{interr}((s\vec{s}_1, s\vec{s}_2), (s\vec{s}'_1, s\vec{s}'_2), (\vec{\tau}_1, \vec{\tau}_2)) T_{interr} = \\
 & \exists i\vec{s}s_1, \vec{t}. \left[\begin{array}{l} \wedge s\vec{s}_1.CS \neq \emptyset \wedge N_{micro}^1(s\vec{s}_1, i\vec{s}s_1, \vec{t}) \wedge higher_pri(\vec{t}, pri(enabled_trans(s\vec{s}_1, T_{interr}))) \\ \wedge N_{micro}^1(s\vec{s}_1, s\vec{s}'_1, \vec{\tau}_1) \wedge s\vec{s}'_2 = update(s\vec{s}_2, \vec{\tau}_1) \\ \wedge \vec{\tau}_2 = \emptyset \wedge higher_pri(\vec{\tau}, \vec{t}) \end{array} \right] \quad (* \text{ component 1 steps } *) \\
 & \vee \\
 & \exists \tau. \left[\begin{array}{l} \wedge \tau \in pri(enabled_trans(s\vec{s}_1, T_{interr})) \wedge (\forall i\vec{s}s_1, \vec{t}. N_{micro}^1(s\vec{s}_1, i\vec{s}s_1, \vec{t}) \implies higher_pri(\{\tau\}, \vec{t})) \\ \wedge s\vec{s}'_1 = update(s\vec{s}_1, \tau) \Big|_{\emptyset}^{CS} \wedge \vec{\tau}_1 = \emptyset \wedge \vec{\tau}_2 = \emptyset \\ \wedge s\vec{s}'_2 = update(s\vec{s}_2, \tau) \Big|_{ent_comp(\tau)}^{CS} \Big|_{n_states_his(s\vec{s}_2, \tau)}^{CS_h} \Big|_{n_ext_ev_his(s\vec{s}_1, \tau)}^{EE_h} \end{array} \right] \quad (* \text{ transition to component 2 } *) \\
 & \vee \quad (* \text{ symmetric cases of two above replaced 1 with 2 and 2 with 1 } *)
 \end{aligned}$$

Figure 5: Semantics of interrupt semantics for micro-steps

Composition Operators

- parallel
- interleaving
- sequence
- choice
- synchronization:
 - environmental
 - rendezvous
- interrupt

Today's Talk

Template Semantics

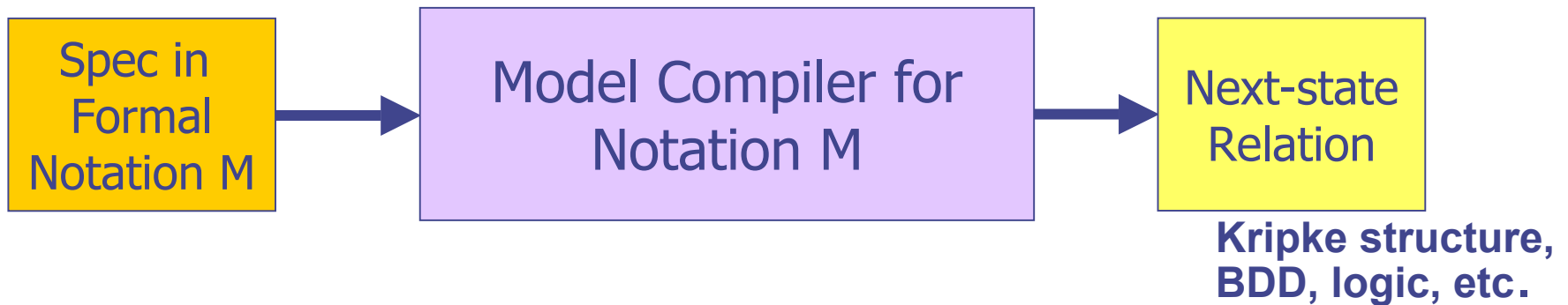
- Template definitions
- Template parameters
- Step semantics
- Composition operators

Generating Analyzers from Template Definitions

Metro

Idea: To generate model compilers from notations' semantics.

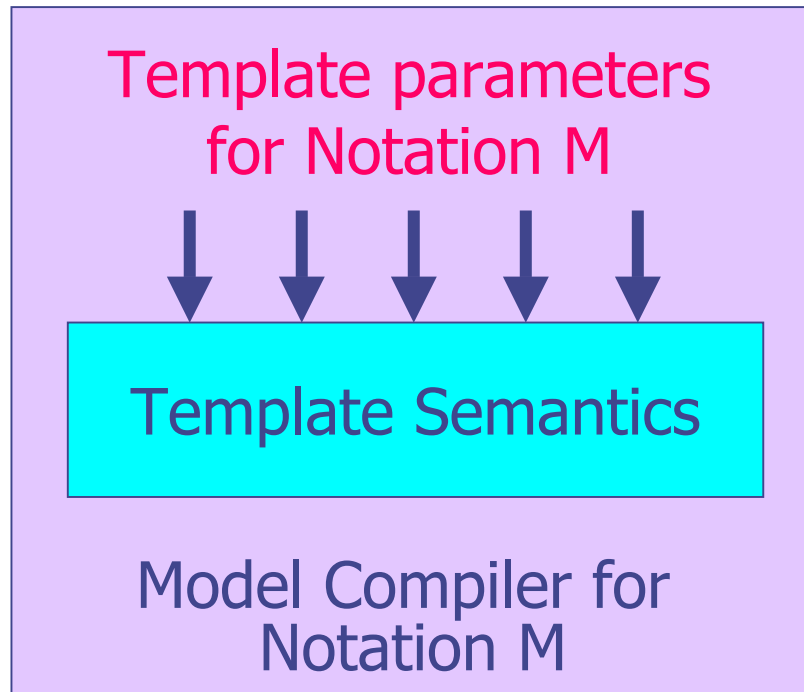
A **model compiler** *compiles* a specification into a more primitive representation, according to the notation's computation *model*.



Metro

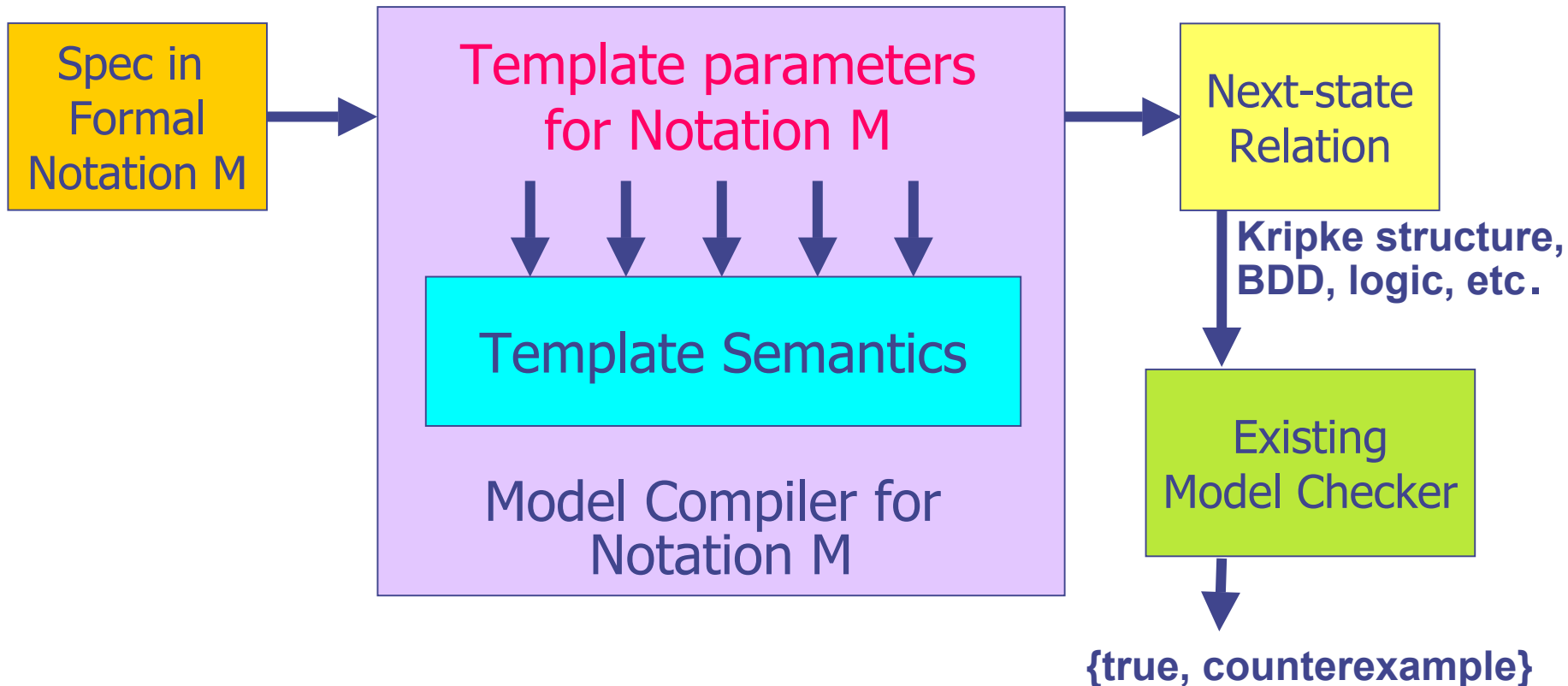
An instantiated template is a model compiler.

It defines a notation's semantics in terms of allowable execution steps.



Metro

When the template's definitions are applied to a specification, it generates a transition-representation that is suitable for analysis.



Summary

We have developed a template approach to defining the operational semantics of model-based notations

- The result is a succinct method of describing the semantics of a specification notation
- Makes it easier to understand and to compare notations
- Makes it easier (possible) to compile specifications in a representation that is more suitable for automated analysis

Current Status

- We have defined the generic template definitions
enabled-transition, apply, init, micro-step, macro-step
- We have defined template parameters and composition operators for several popular notations
various statecharts variants, RSML, SCR, SDL88, Petri-Nets, process algebras
- We have implemented a vertical slice of the Metro model-compiler generator to handle Basic State Transitions
enabled-transition, apply, macro-step, interleaving composition,

Future Work

- Continuing the implementation of Metro model-compiler generator
- Applying Metro to more sophisticated notations
 - Z?
 - Abstract State Machines?
 - Paderborn's semantics for UML statecharts?
- Handling multi-notation specifications