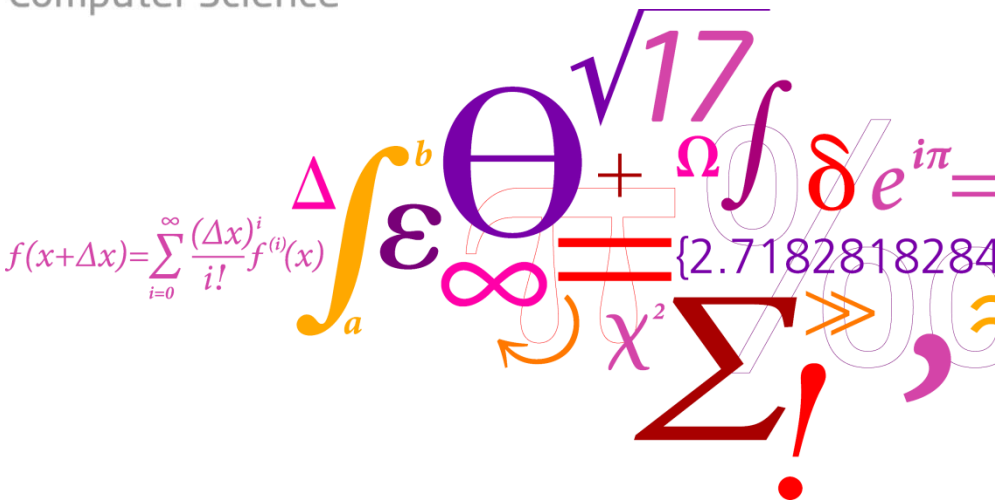
Advanced Topics in Software Engineering (02265)

Ekkart Kindler

DTU Compute
Department of Applied Mathematics and Computer Science



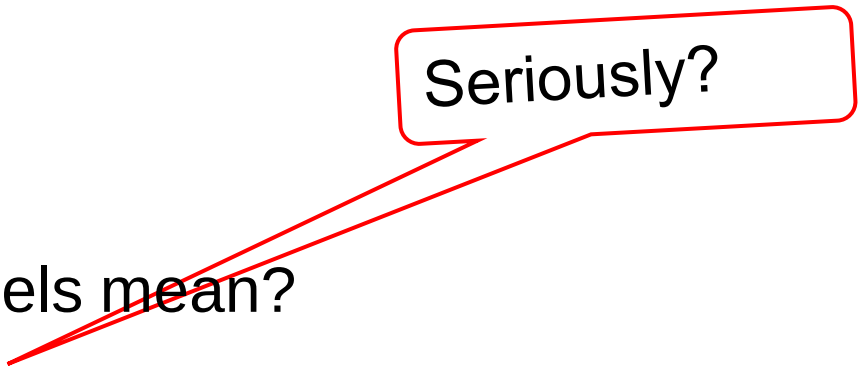
VI. Formalisation and Analysis

1. Motivation

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Questions:

- Why do we use models?
  - Understanding problems / solutions
  - Communication of ideas
  - Code generation / execution
  - Analysis and Verification



- How do we define what models mean?
  - MOF can be defined in itself?!
  - In natural language (typically in English)
  - Mathematics (the ultimate resort in every field)

  In particular, when it comes to behaviour models, MOF is not (yet?) powerful enough to define it.

Motivation

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Questions:

- How do we make sure that the models are correct?
  - Analyse the models (and the state space)
  - ”Formal methods”: all kinds of clever techniques to analyse and verify models **efficiently** (avoiding exploring all states explicitly, representing sets of states symbolically, ...)

- How can we be sure the generated code is correct?
  - Define the semantics of both the model and the code
  - Verify that the code generator preserves them

Motivation

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

As long as we cannot express the meaning of models fully in MOF:

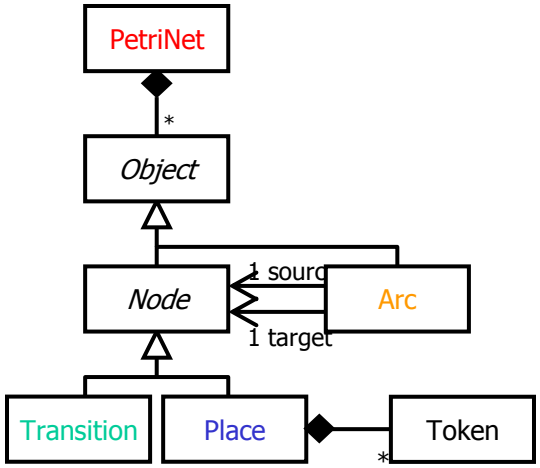
- We need to be able to formalize the syntax and **the semantics** in mathematics

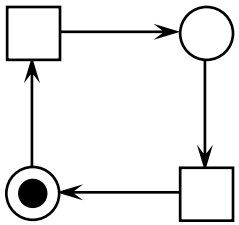
In our case, mainly the behaviour.

A thorough study would be separate courses: “Formal Methods” “Semantics”, “Verification”, “Model checking”

Here, we confine ourselves to a systematic example: Petri nets

Example: Petri nets





Definition 1 (Petri net)

A *Petri net* $N = (P, T, F)$ consist of two disjoint sets P and T and a relation $F \subseteq (P \times T) \cup (T \times P)$.

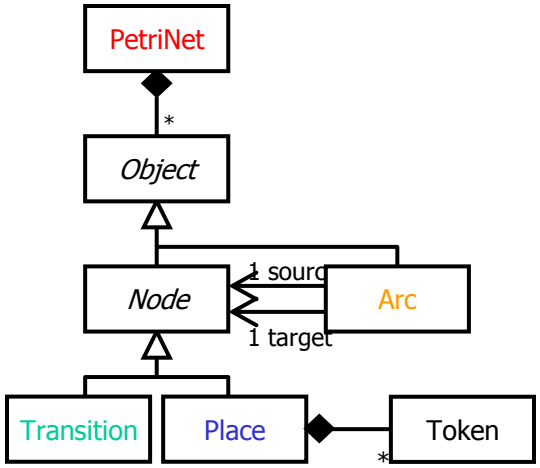
The elements of P are called the *places* of N, the elements of T are called the *transitions* of N, and the elements of F are called the *arcs* of N.

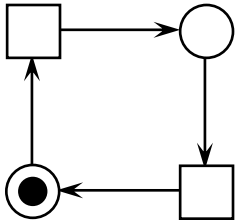
The relation F is also called the *flow-relation* of N.

Sometimes, one requires P and T to be finite sets.

Formalising (abstract) syntax

Example: Petri nets





Definition 2 (Marking of a Petri net)

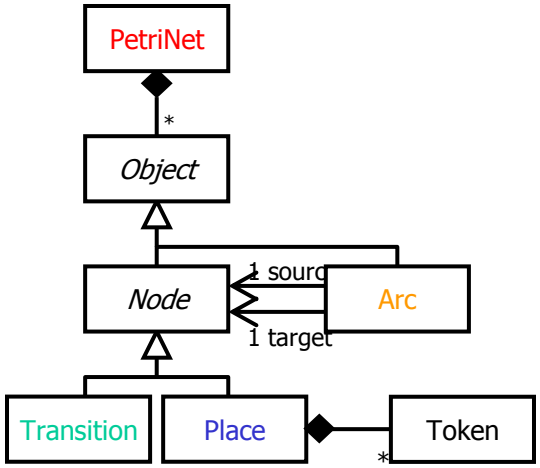
Let $N = (P, T, F)$ be a Petri net.
A *marking* of N is a mapping
$m: P \rightarrow IN$.

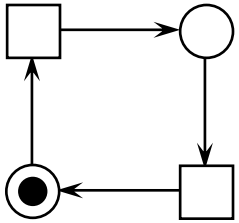
Standard symbol for the set of natural numbers: 0, 1, 2, 3, …

Formalising (abstract) syntax

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

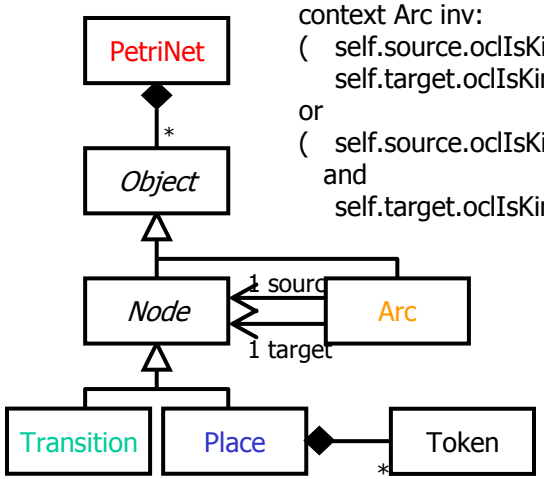
Example: Petri nets



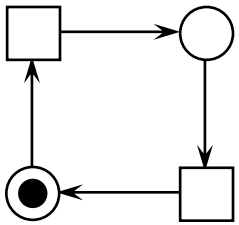


Formalising (abstract) syntax

Example: Petri nets



context Arc inv:
(self.source.oclIsKindOf(Place) and
self.target.oclIsKindOf(Transition))
or
(self.source.oclIsKindOf(Transition)
and
self.target.oclIsKindOf(Place))



Any differences between the meta-model and the mathematics?

Definition 1 (Petri net)

A *Petri net* $N = (P, T, F)$ consist of two disjoint sets P and T and a relation $F \subseteq (P \times T) \cup (T \times P)$.

Definition 2 (Marking of a Petri net)

Let $N = (P, T, F)$ be a Petri net.
A *marking* of N is a mapping
$m: P \rightarrow IN$.

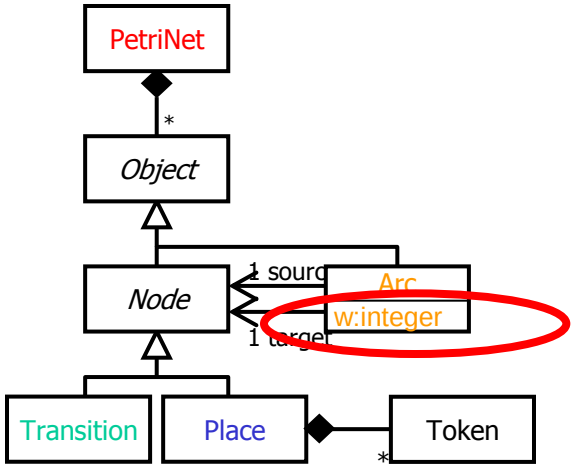
Definition 3 (Petri net system)

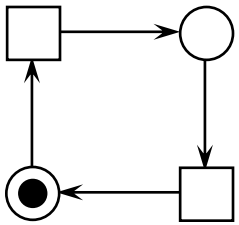
Let N be a Petri net and let m_0 be a marking of N. Then, we call
$\Sigma = (N, m_0)$ a *Petri net system*.

Formalising (abstract) syntax

Variation of Petri nets (maybe the most typical form of Petri nets)

Example: Place/Transition system





Definition 1 (Petri net)

A *Petri net* $N = (P, T, F)$ consist of two disjoint sets P and T and a relation $F \subseteq (P \times T) \cup (T \times P)$.

Definition 2 (Marking of a Petri net)

Let $N = (P, T, F)$ be a Petri net.
A *marking* of N is a mapping $m: P \rightarrow IN$.

Definition 4 (Place/Transition system)

Let $N = (P, T, F)$ be a Petri net, let m_0 be a marking of N *and* $W: F \rightarrow IN \setminus \{0\}$.
Then, we call $\sum = (N, W, m_0)$ a *Place/Transition-system* (P/T-system).

Observations

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

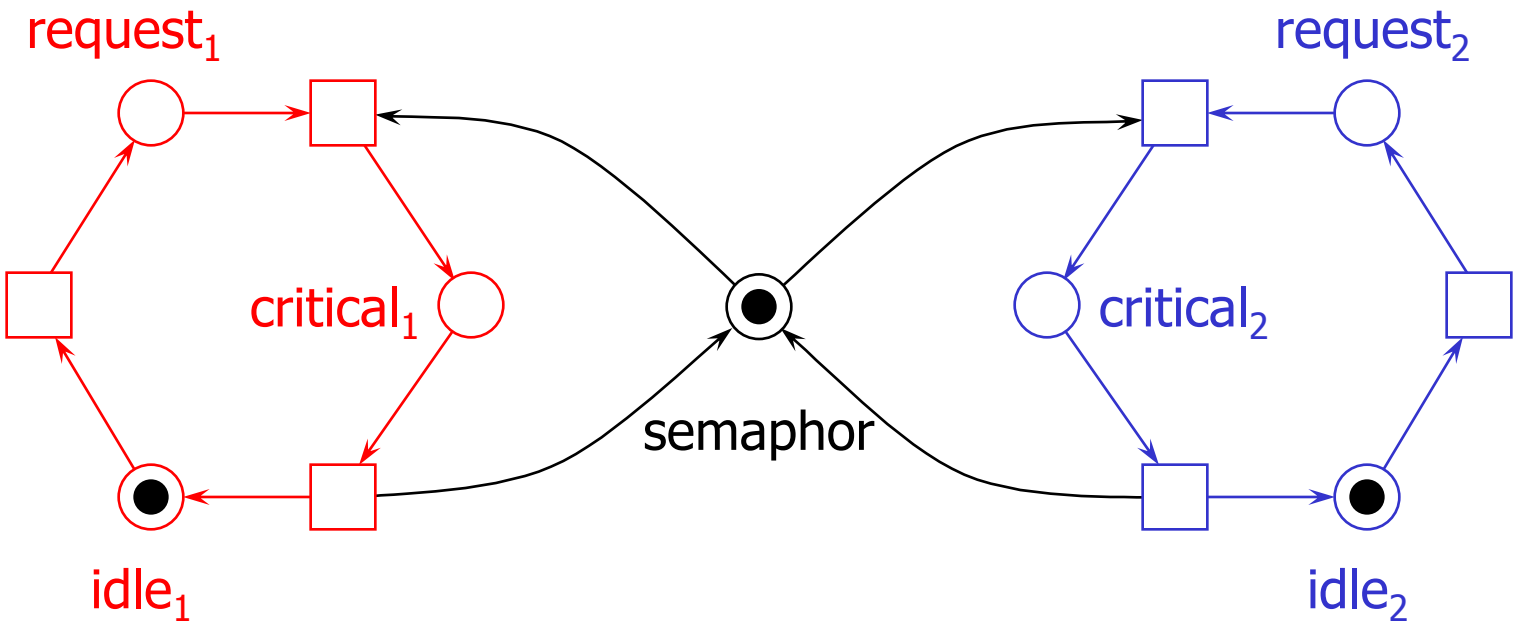
- Nodes of a formalism represented as sets
  - different sets for different kinds of nodes
  - different kind: disjointness of sets
- Arcs between nodes as a relation
  - Constraints in form of a restriction
- Labels as mappings

Conceptually, tokens are labels of places (number of tokens).

- Definitions systematically build on each other (kind of modular)

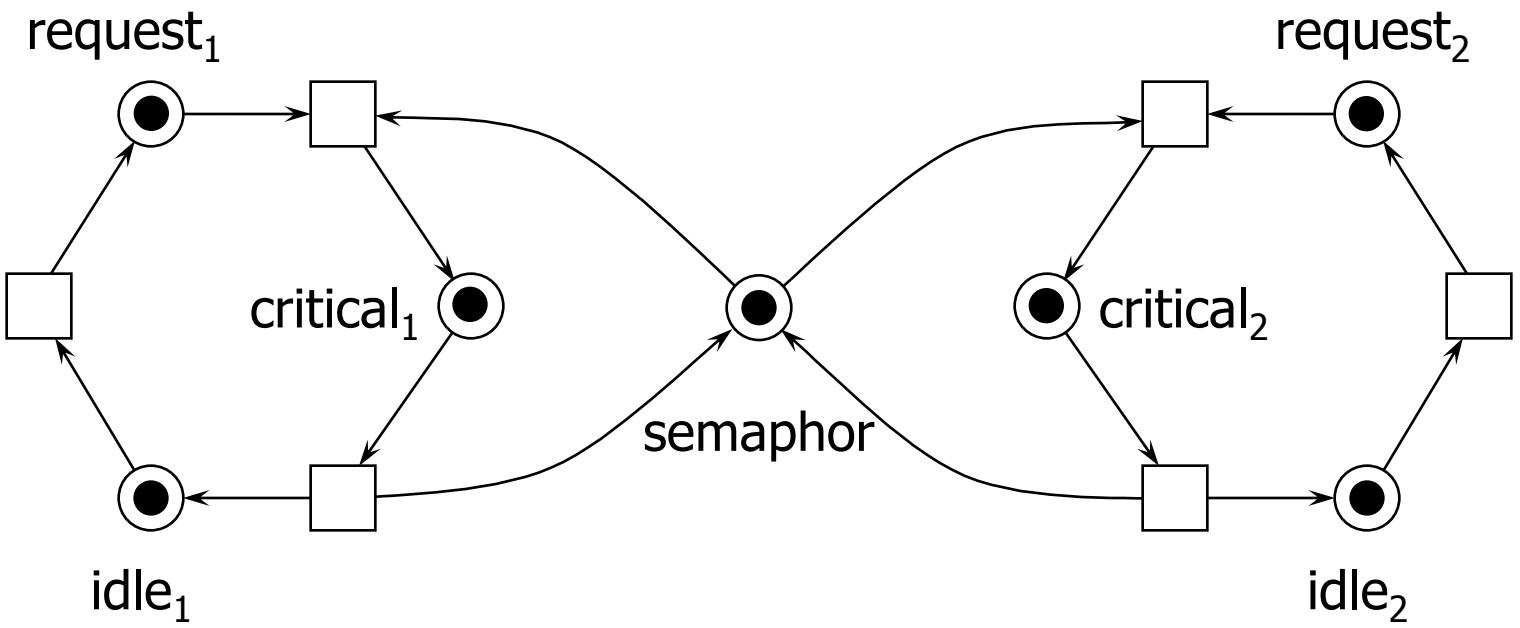
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Example



Firing rule

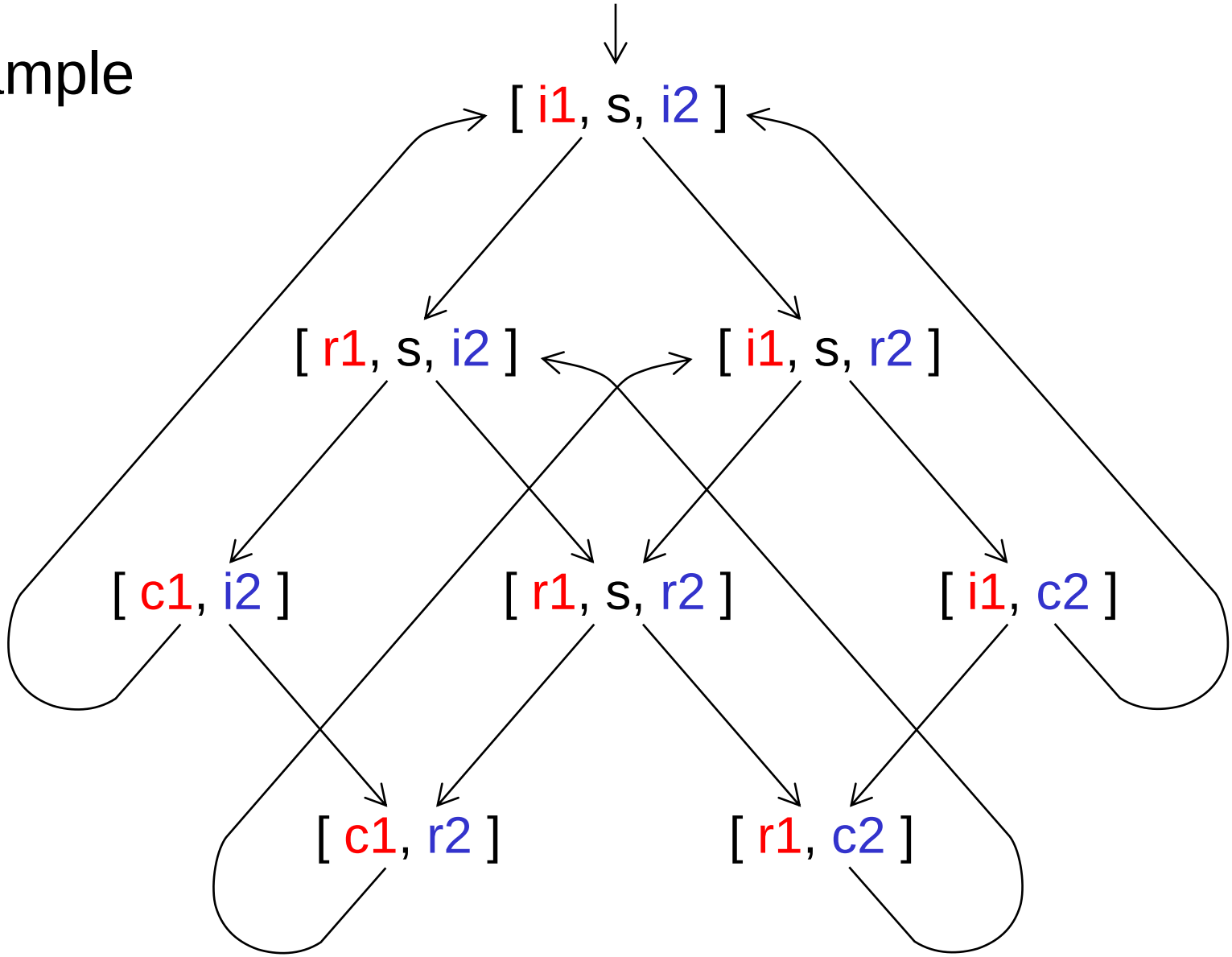
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



Reachabilitygraph

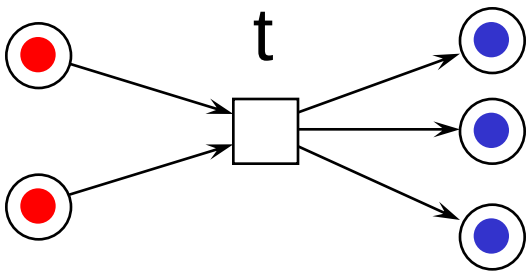
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Example



DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Example: Petri nets



Definition 5 (Markings ${}^{-}t$ and t^{+})

Let $N = (P, T, F)$ be a Petri net and $t \in T$ be a transition.

The marking ${}^{-}t : P \to IN$ is defined by:

$${}^{-}t(p) = 1, \text{ if } (p,t) \in F, \text{ and}$$
$${}^{-}t(p) = 0, \text{ if } (p,t) \notin F$$

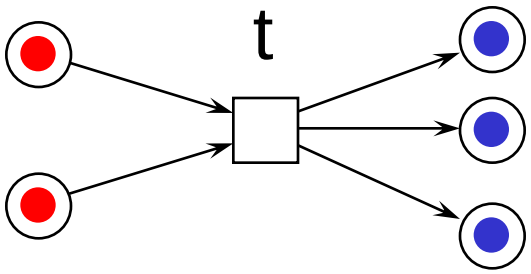
The marking $t^{+} : P \to IN$ is defined by:

$$t^{+}(p) = 1, \text{ if } (t,p) \in F, \text{ and}$$
$$t^{+}(p) = 0, \text{ if } (t,p) \notin F$$

The relations ≥, ≤, and the operations + and - carry over to markings (pointwise).

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Example: Petri nets



Definition 6 (Firing rule)

Let $N = (P, T, F)$ be a Petri net, $t \in T$ be a transition, and m be a marking of N.

A transition t is *enabled* in marking m, if $m \geq {}^{-}t$.

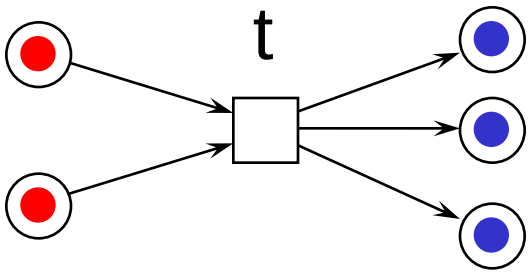
Then, we write $m \xrightarrow{t}$

If the transition t is enabled in m, the transition can fire, which results in the successor marking $m' = (m - {}^{-}t) + t^{+}$.

Then, we write $m \xrightarrow{t} m'$.

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Example: Petri nets



Definition 7 (Reachable markings)

Let $\Sigma = (N , m_0)$ be a Petri net system.

The set of *reachable markings* R_Σ of Σ is defined as the least set, such that

- $m_0 \in R_\Sigma$
- if $m \in R_\Sigma$ *and there exists a transition t of N and a marking m' such that* $m \xrightarrow{t} m'$, *then also* $m' \in R_\Sigma$

This **inductive definition**, actually, "defines" an algorithm to calculate all reachable states (if the set is finite).

Observations

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

The way of defining the behaviour very much depends on the formalism, but

- Typically there is some notion of state (markings in our example)
- There is one (or more) initial state
- There is a transition relation $m \xrightarrow{t} m'$

In Petri nets these are the transitions; in other formalisms, it could be events or something more complex.

In our example, we just defined the reachable markings. The reachablility graph would contain also these transitions.

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

The inductive definition of the reachable states gives an algorithm for computing it (in the finite case):

R:= { } // set of already found reachable states

U:= { m_0 } // set of states that are yet undealt with

while $U \neq \{\ \}$ do

  select any $m \in U$

  U:= $U \setminus \{ m \}$

  R:= $R \cup \{ m \}$

  for each m' with $m \rightarrow m'$ do

    U:= $U \cup \{ m' \}$

result is R

State space generation

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

The inductive definition of the reachable states gives an algorithm for computing it (in the finite case):

$R := \{\}$ // set of already found reachable states

$U := \{ m_0 \}$ // set of states that are yet undealt with

while $U \neq \{\}$ do

  select any $m \in U$

  $U := U \setminus \{ m \}$

  $R := R \cup \{ m \}$

  for each m' with $m \rightarrow m'$ do

    $U := U \cup \{ m \}$

result is R

Warning: This algorithm does not terminate—even when there are only finitely many reachable states!

Why?

As soon as there is some cycle in the reachability graph, this algorithm does not work!

State space generation

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

The inductive definition of the reachable states gives an algorithm for computing R (and U):

$R = \{\ \}$

$U = \{\ m_0\ \}$

Could be lifo, fifo, random; this does not influence the result (in the finite case), only the order in which it is computed.

while $U \neq \{\ \}$ do

  select any $m \in U$

  $U := U \setminus \{\ m\ \}$

  $R := R \cup \{\ m\ \}$

  for each m' with $m \rightarrow m'$ do

    if $m' \notin R$ then $U := U \cup \{\ m'\ \}$

result is R

Now, the algorithm terminates, if there are only finitely many reachable states.

How about efficiency?

State space generation

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Where are the bottlenecks?

$R = \{\ \}$
$U = \{\ m_0 \}$
while $U \neq \{\ \}$ do
  select any $m \in U$
  $U := U \setminus \{\ m \}$
  $R := R \cup \{\ m \}$
  for each m' with $m \rightarrow m'$ do
    if $m' \notin R$ then $U := U \cup \{\ m \}$

result is R

Sometimes, identifying which transitions are possible requires some effort.

For Petri nets, this can be done much more efficiently if we remember which transitions have been enabled in m in order to calculate the ones that are enabled in m'.

Similar techniques exist for other formalisms. The details very much depend on the formalism.

R can be huge! Iterating over it for checking $\notin$ is a bad idea!
Use hash function or another good idea.

Checking properties on the fly

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

$R = \{\ \}$
$U = \{\ m_0 \}$
while $U \neq \{\ \}$ do
  select any $m \in U$
  $U := U \setminus \{\ m \}$
  $R := R \cup \{\ m \}$
  for each m' with $m \rightarrow m'$ do
    if $m' \notin R$ then $U := U \cup \{\ m \}$

result is R

Some properties of the system can be checked on the fly:
→ invariants
→ deadlocks

5. Model checking

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Here we consider model checking (or some techniques from model checking) as an example for the systematic analysis of the state space.

Today, model checking is a field of its own and could cover a full 10 ECTS-point course; here, we just give an overview.

5.1. Terminology

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Model Checking
- Validation and Verification
- Reactive System

Model Checking

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Model checking is a
technology for
the fully automatic
verification of
reactive systems
with a finite state space.

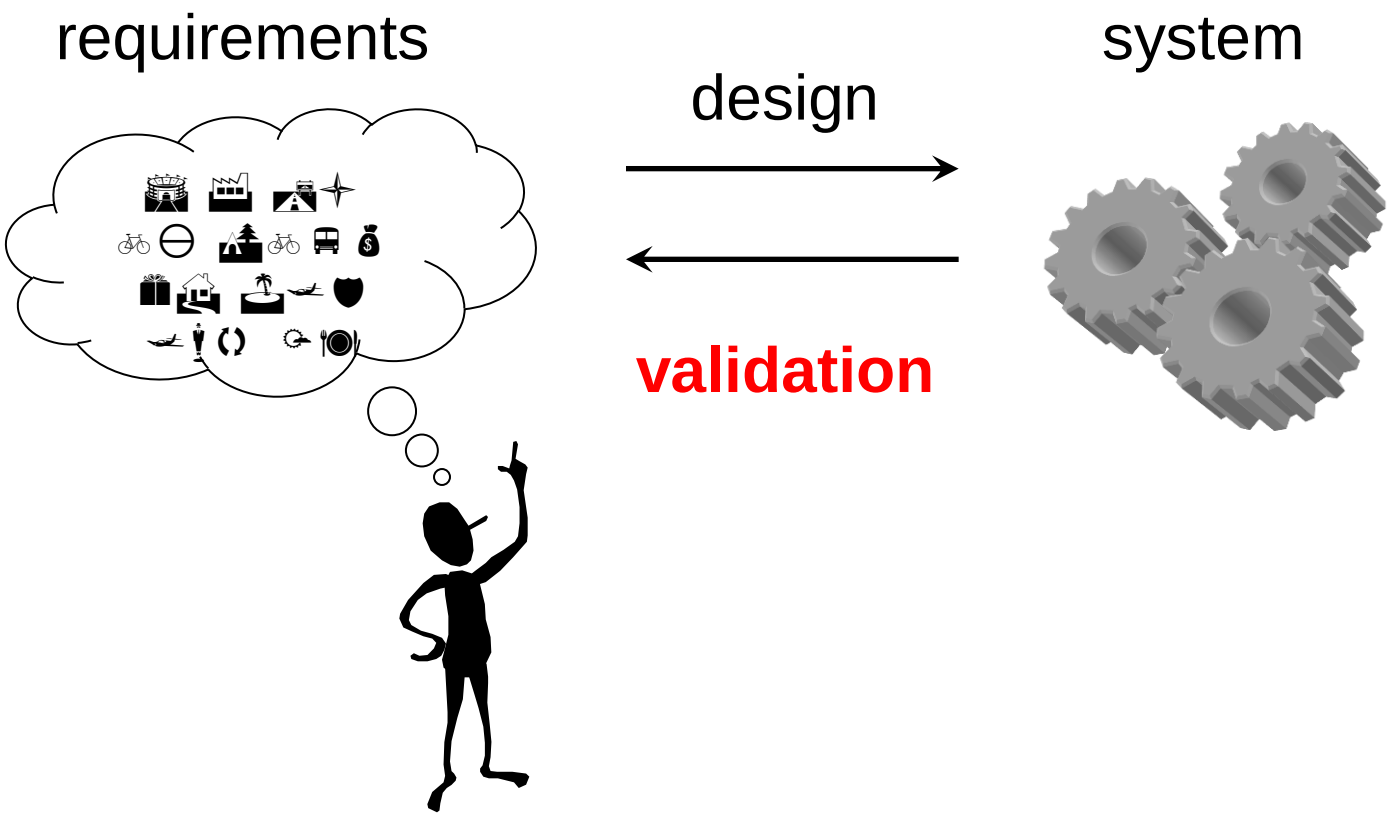
Which should typically be quite small.
Some advanced techniques can even deal with infinite state spaces.

Terms

- **Technology**
  - principle
  - method
  - concept
  - notation
  - tool
- **System**
  - **reactive** vs. transformational
  - model
- Validation
  - requirements
  - specification
  - simulation
  - test
  - **verification**
    - deductive
    - model based

Validation

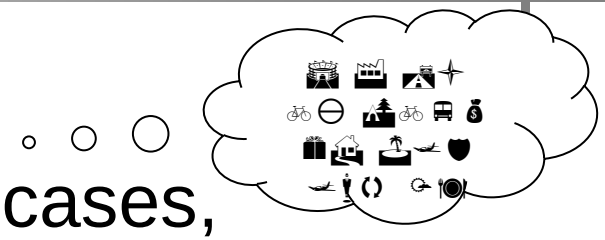
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



Question: Does the system do what it should do?

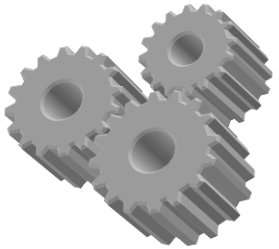
Validation

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



Problems:

- requirements are informal in most cases, imprecise, incomplete, inconsistent, …
- systems can be very complex



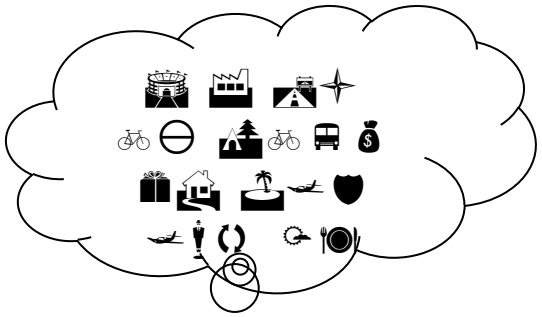
- designing and building systems is very expensive
- the later a flaw is detected the higher the costs to repair it

Validation

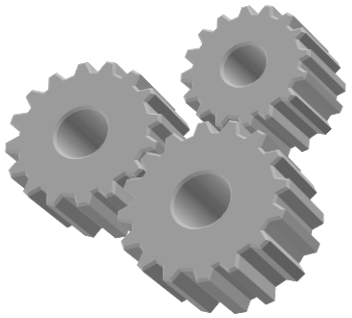
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

requirements

system



validation

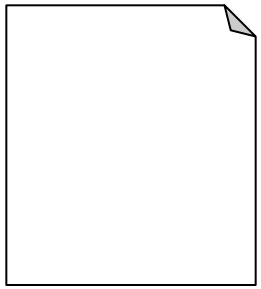


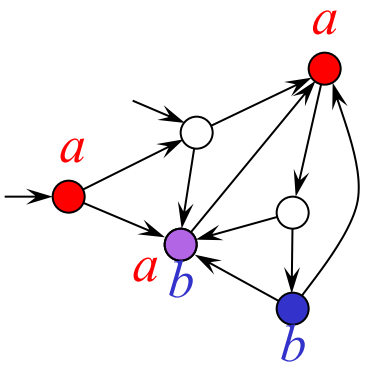
formali-
sation

validation

abstraction

refinement /
implementation





verification

or code
generation!

(formal)
specification

model

Validation

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Remarks:

- most requirements are informal
- validation is an **inherently informal** process
- checking whether a specification captures the requirements is inherently informal

- verification is a formal process (automatic in some cases) that can partially help with validation

Transformational System

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- accepts some **input**
- makes some **calculations**
- returns a **result**

In particular:

- terminates always (resp. should terminate)
- no user interaction possible
  (after the input was accepted)

Reactive System

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- reacts permanently to input
- can output results any time
  (dependent on the input)

In particular:

- is interactive (could even be active or proactive)
- does not terminate (normally)
- reactive systems do not „calculate a function“

Reactive vs. transformational

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Information systems are reactive (in most cases)

The classical notions of algorithm and computation are defined from the transformational system's point of view

Reactive systems have transformational components in most cases

Model Checking

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Model checking is tailored to the verification of reactive systems

- special notations for „reactive properties“ (temporal logics)
- abstraction from transformational parts (and often from data)
- appropriate for cyclic behaviour

- but on a high level of abstraction only

Summary

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Model checking is a
technology for
the fully automatic
verification of
reactive systems
with a finite state space.

5.2. Main Concepts and Ideas

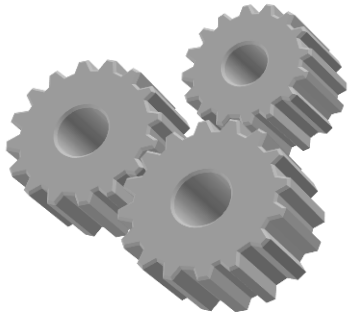
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Kripke structures (defining the system/model)
- CTL (specifying the properties)
- algorithms (only basic idea)
- complexity

Systems and Requirements

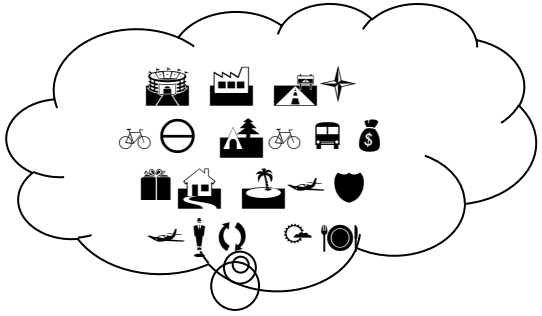
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

system



meets

requirements

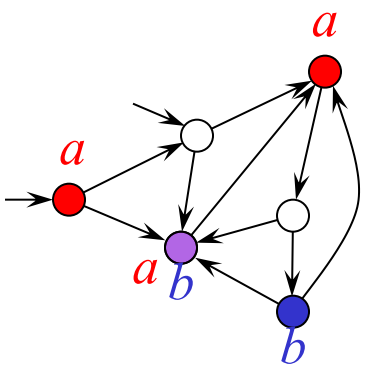


Model und Specification

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

model M

specification A



$\models$

$\mathbf{AG}\,(\,a \Rightarrow \mathbf{AF}\,b\,)$

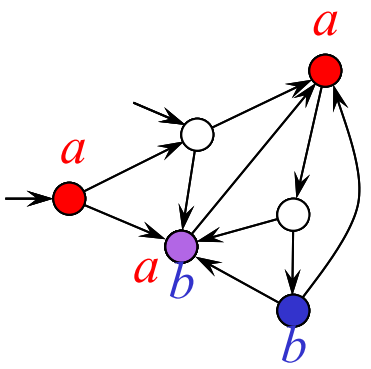
Kripke structure

Computation **T**ree **L**ogic (**CTL**)

Kripke Structure

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

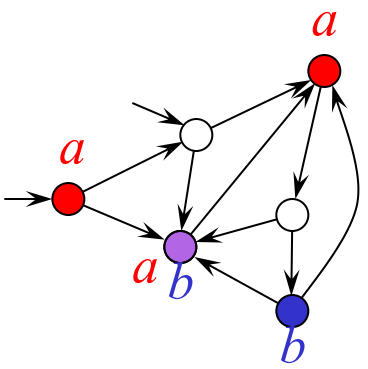
A **Kripke structure** consists of

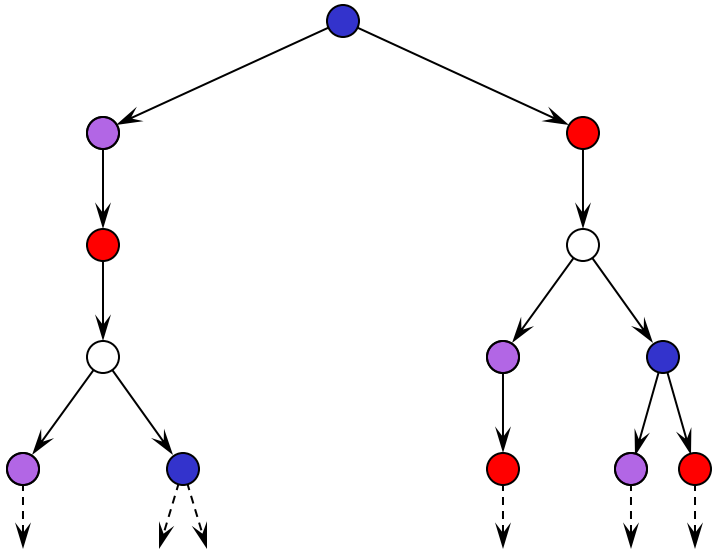


- a set of **states**,
- with distinguished **initial states**,
- a (total) **transition relation** and
- a **labelling** of states with a set of **atomic propositions**.

Behaviour

The **behaviour** at a state can be represented as a **computation tree**:





Note that all paths are infinite!

That is a consequence of the totality of the transition relation.

CTL-Formulas

$a, b, \ldots$

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

$. \wedge . , . \vee . , \neg . , \ldots$

$\mathbf{EX}\ . , \mathbf{EG}\ . , \mathbf{E}[\ . \ \mathbf{U}\ . \ ], \ldots$

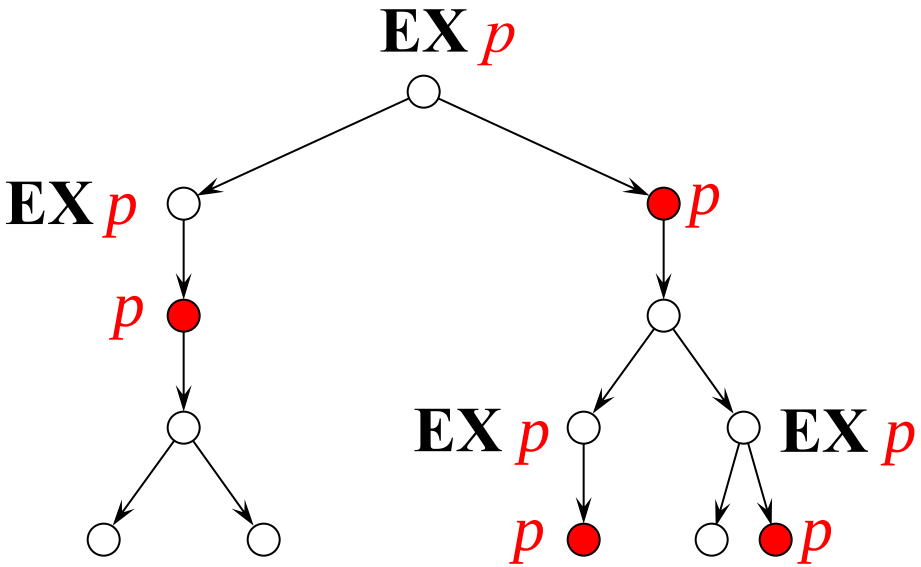
CTL-formulas are inductively defined:

- atomic propositions are CTL-formulas
- CTL-formulas combined with a Boolean operator are CTL-formulas
- CTL-formulas combined with temporal operators are CTL-formulas

Exists neXt: EX p

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

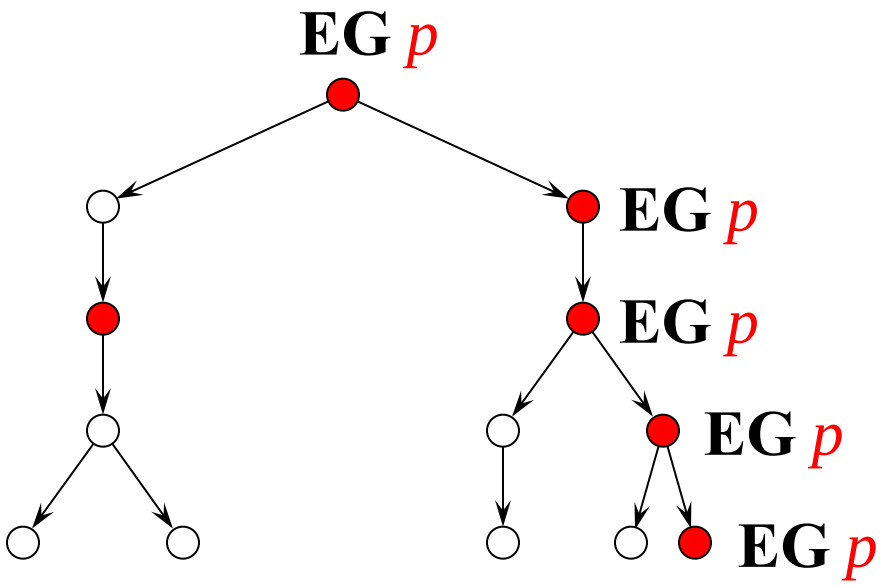
there exists an (immediate) successor in which p holds true:



Exists Globally: EG p

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

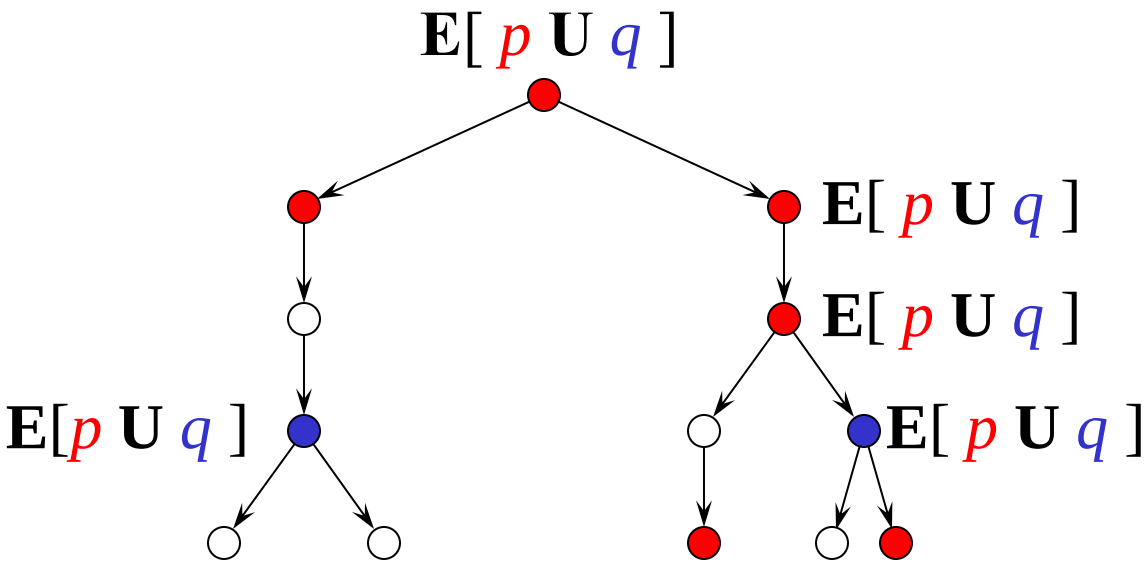
there exists an infinite path on which p holds in each state:



Exists Until: $\mathbf{E}[\, p \;\mathbf{U}\; q \,]$

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

there exists a reachable state in which b holds true, and up to this state p holds true:



Abbreviations

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

$\mathbf{AX}\ p \equiv \neg\, \mathbf{EX} \neg\, p$

for all immediate successors, p holds true

$\mathbf{EF}\ p \equiv \mathbf{E}\ [\ \mathit{true}\ \mathbf{U}\ p\ ]$

in some reachable state, p holds true

$\mathbf{AG}\ p \equiv \neg\, \mathbf{EF} \neg\, p$

in all reachable states, p holds true

$\mathbf{AF}\ p \equiv \neg\, \mathbf{EG} \neg\, p$

on each path, there exists a state in which p holds true

Dualities

System “meets” formula

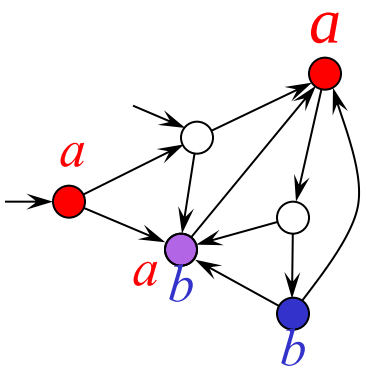
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

A CTL-formula **holds** for a Kripke structure if the formula holds in each initial state.

Example

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

model M $\models$ specification p



$$\models \quad \mathbf{AG}\,(\,a \Rightarrow \mathbf{AF}\,b\,)$$

How do we prove it?

Algorithms

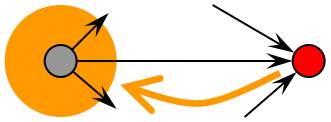
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

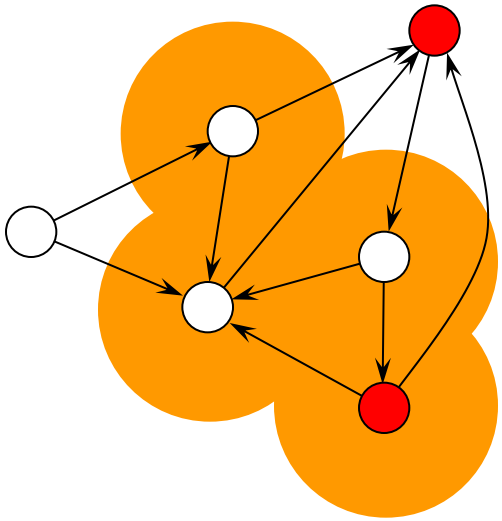
For each sub-formula, we inductively calculate the **set of states**, in which this sub-formula is true:

- atomic propositions ☑
- Boolean operators ☑
- temporal operators

„Algorithm“ for $\mathbf{EX}\ p$

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler





Given:
The set of states in which p holds: S_p

Wanted:
The set of states in which $\mathbf{EX}\ p$ holds: $S_{\mathbf{EX}\ p}$

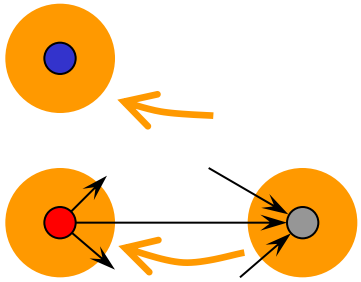
We also write $\mathbf{EX}(S_p)$ for $S_{\mathbf{EX}\ p}$

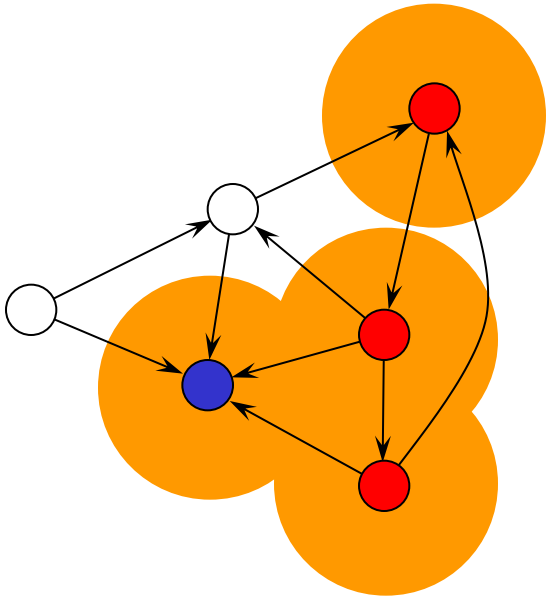
Algorithm for $\mathbf{E}[\, p \;\mathbf{U}\; q \,]$

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

As stated here, this algorithm is quite inefficient. There are more efficient ways to do this.

But, even this inefficient algorithm turns out to be quite efficient when used with the right data structure (ROBDDs, see next lecture).





Given: S_p und S_q

Wanted: $S_{\mathbf{E}[\, p \;\mathbf{U}\; q \,]}$

$$S_0 = \varnothing$$

$$S_1 = S_q \cup (S_p \cap \mathbf{EX}(S_0))$$

$$S_2 = S_q \cup (S_p \cap \mathbf{EX}(S_1))$$

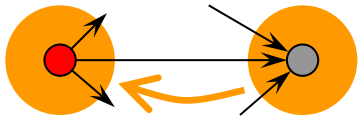
...

$$S_{i+1} = S_q \cup (S_p \cap \mathbf{EX}(S_i))$$

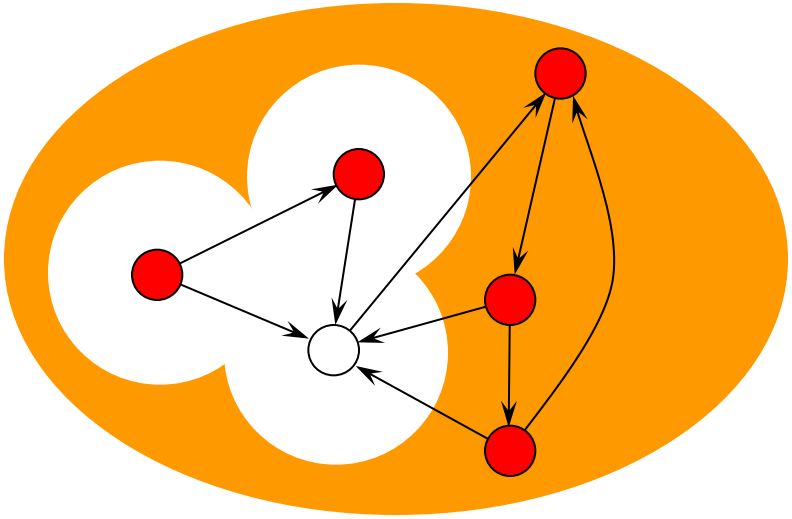
until $S_{i+1} = S_i = S_{\mathbf{E}[\, p \;\mathbf{U}\; q \,]}$

Algorithm for $\mathbf{EG}\ p$

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



Ditto



Given: S_p
Wanted: $S_{\mathbf{EG}\,p}$

$$S_0 = S$$

$$S_1 = S_p \cap \mathrm{EX}(S_0)$$

$$S_2 = S_p \cap \mathrm{EX}(S_1)$$

...

$$S_{i+1} = S_p \cap \mathrm{EX}(S_i)$$

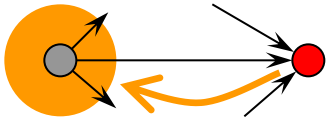
until $S_{i+1} = S_i = S_{\mathbf{EG}\,p}$

Algorithms Summary

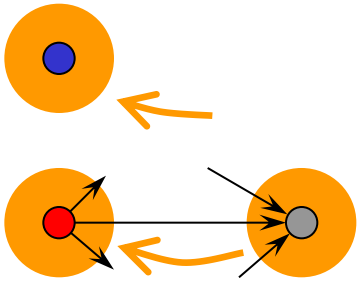
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

CTL model checking ~ marking algorithm + iteration

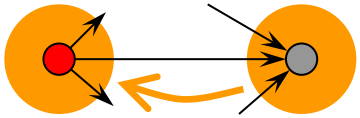
- **EX** p



- **E**[p **U** q]



- **EG** p



Complexity

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

When implemented in an efficient way, the marking algorithm for each operator is linear in the number of states of the system:

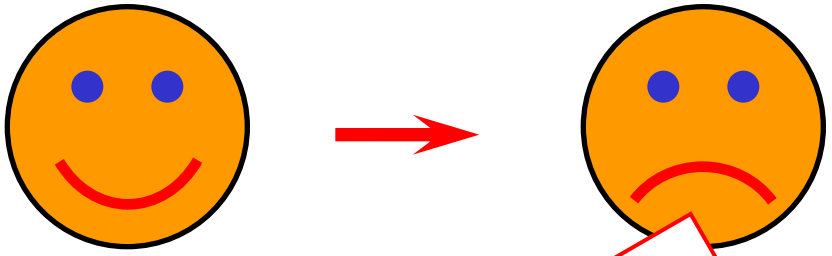
$$O(|M| \cdot |p|)$$

size of the model

size of the formula

Complexity

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



When implemented in an efficient way, th… algorithm for each operator is linear … number of states of the system:

$$O(|M|$$

state space explosion