How to build a Model Cheker?

Etienne Renault & Alexandre Duret-Lutz

2016, February 16th

https://www.lrde.epita.fr/~renault/teaching/imc/

Foreword

Today we will build a model checker !

At the end of the day, you will be able :

Foreword

Today we will build a model checker !

At the end of the day, you will be able :

- to express properties using LTL (see previous lesson)

Foreword

Today we will build a model checker !

At the end of the day, you will be able :

- to express properties using LTL (see previous lesson)
- to understand how a (basic) model-checker works

Foreword

Today we will build a model checker !

At the end of the day, you will be able :

- to express properties using LTL (see previous lesson)
- to understand how a (basic) model-checker works
- to create a model, i.e an abstract representation of a system

Foreword

Today we will build a model checker !

At the end of the day, you will be able :

- to express properties using LTL (see previous lesson)
- to understand how a (basic) model-checker works
- to create a model, i.e an abstract representation of a system
- to check if the model meets the specification, i.e. if the system behaves as expected

Foreword

Today we will build a model checker !

At the end of the day, you will be able :

- to express properties using LTL (see previous lesson)
- to understand how a (basic) model-checker works
- to create a model, i.e an abstract representation of a system
- to check if the model meets the specification, i.e. if the system behaves as expected

What is a system ?

All theses pictures are under CreativeCommons

Why a model is required ?

The following server-like snippet can be considered as a system.

```
unsigned received_ = 0;
while (1)
{
  accept_request();
  received_ = received_ + 1;
  reply_request();
}
```

How many configurations for such a program ?

We have 2 unsigned variables (received_ + Program Counter).
In the worst case : $(2^{32} - 1)^2$

What is a model ?

Real systems have hundreds of thousands variables !

Since model checker may explore all these configurations, we must reduce the memory complexity.

A model is an abstract representation of the system

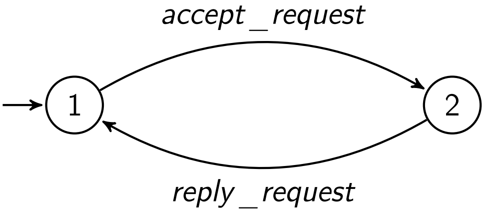
- A model has less variables than the real system
- A model has less *configurations* than the real system
- A model mostly focuses on behaviors and interactions
- A model has **a finite number of variables**, i.e. no dynamic allocations

How to represent a model ?

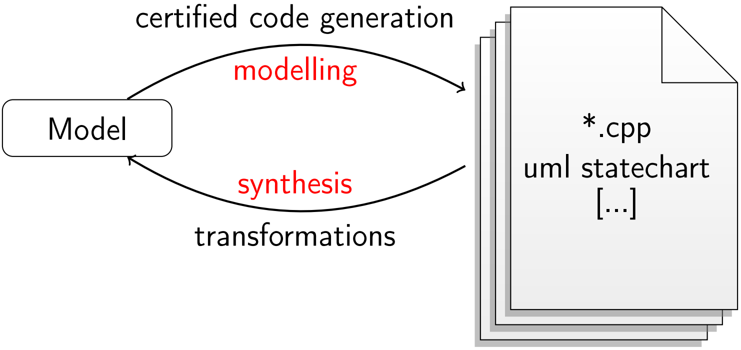
Each component of the system can be represented like an finite state automaton

- possible only since there is a finite number of finite size variables

The previous server-like snippet can then be represented as following :



How to build a model ?



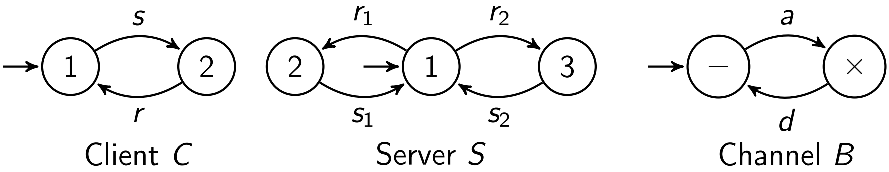
Model formalisms

There are a lot of formalisms :

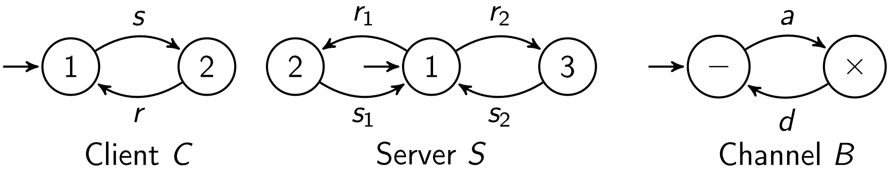
- PetriNet, Fiacre, **DVE**, Promela, AADL, etc.

All are not equivalent but there are all formally specified.

A more realistic example !

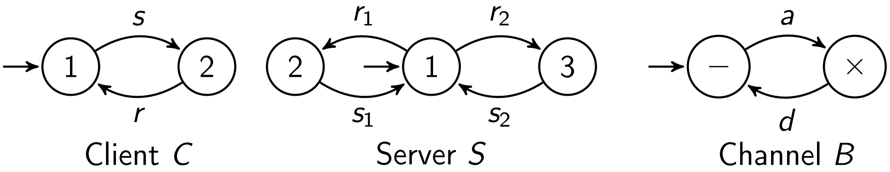


A more realistic example !



1 server, 2 clients, 4 channels

A more realistic example !

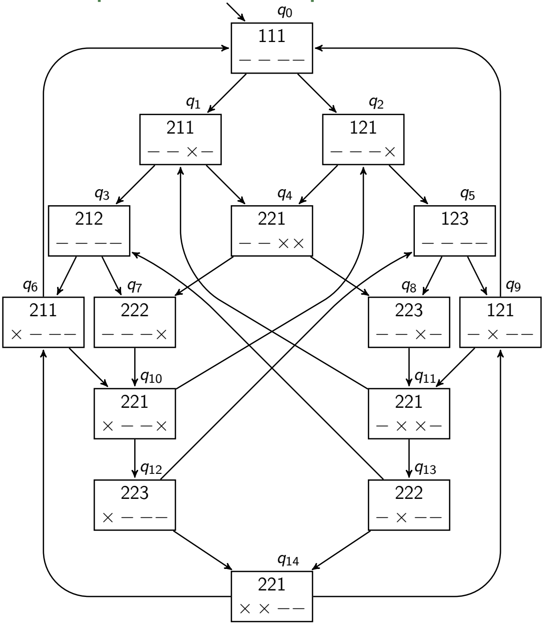


1 server, 2 clients, 4 channels

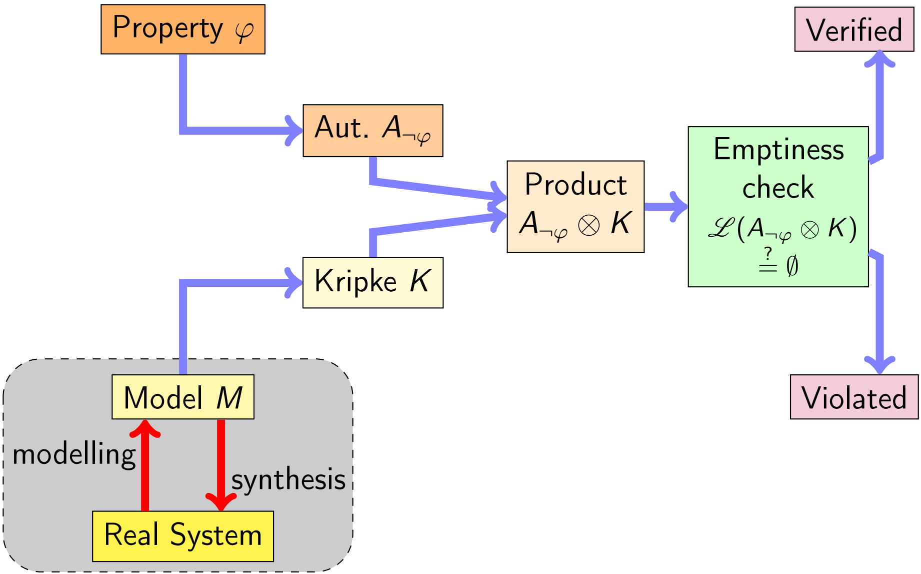
System's synchronization rules $\langle C, C, S, B, B, B, B \rangle$:

(1)	$\langle\ s\ ,\ .\ ,\ .\ ,\ .\ ,\ .\ ,\ a\ ,\ .\ \rangle$
(2)	$\langle\ .\ ,\ s\ ,\ .\ ,\ .\ ,\ .\ ,\ .\ ,\ a\ \rangle$
(3)	$\langle\ r\ ,\ .\ ,\ .\ ,\ d\ ,\ .\ ,\ .\ ,\ .\ \rangle$
(4)	$\langle\ .\ ,\ r\ ,\ .\ ,\ .\ ,\ d\ ,\ .\ ,\ .\ \rangle$
(5)	$\langle\ .\ ,\ .\ ,\ r_1\ ,\ .\ ,\ .\ ,\ d\ ,\ .\ \rangle$
(6)	$\langle\ .\ ,\ .\ ,\ s_1\ ,\ a\ ,\ .\ ,\ .\ ,\ .\ \rangle$
(7)	$\langle\ .\ ,\ .\ ,\ r_2\ ,\ .\ ,\ .\ ,\ .\ ,\ d\ \rangle$
(8)	$\langle\ .\ ,\ .\ ,\ s_2\ ,\ .\ ,\ a\ ,\ .\ ,\ .\ \rangle$

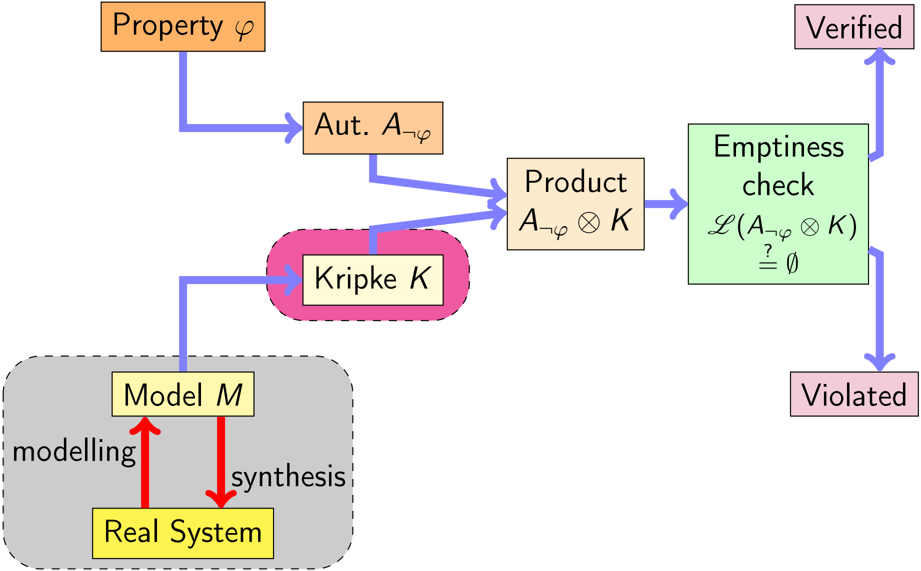
Example's state space



Automata approach for model checking



Automata approach for model checking



Kripke structure

State machine labelled by atomic propositions.

A Kripke structure is a 5 tuple $K = \langle AP, \mathcal{Q}, q^0, \delta, l \rangle$ with

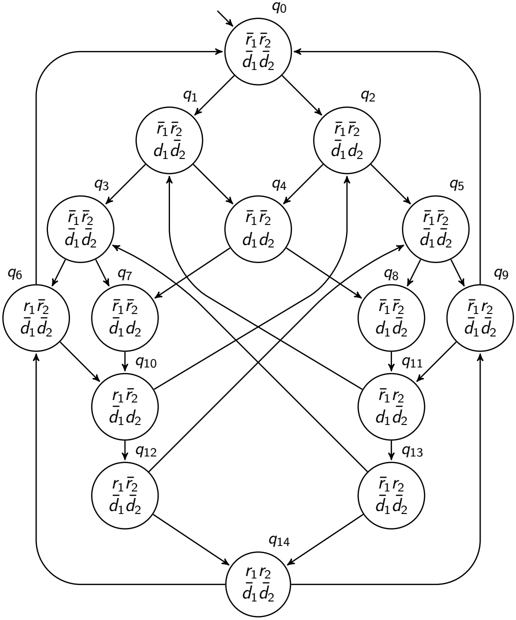
- AP is the set of atomic propositions
- $\mathcal{Q}$ is the finite set of state
- $q^0 \in \mathcal{Q}$ is the initial state
- $\delta : \mathcal{Q} \mapsto 2^{\mathcal{Q}}$ is the transition function that associates successors to a given state
- $l : \mathcal{Q} \mapsto 2^{AP}$ is labelling function that associates atomic propositions to a given state

Atomic propositions for the example

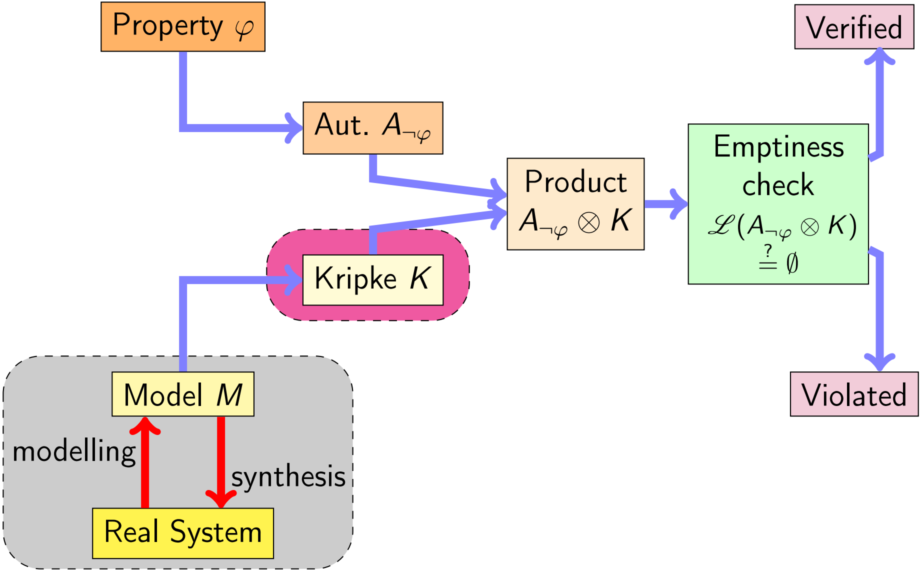
We want to track messages received and sent. Let us define $AP = \{r_1, r_2, d_1, d_2\}$, s.t. :

- r_1 : a response is in progress between the server and the first client
- r_2 : a response is in progress between the server and the second client
- d_1 : a request (d for demand) is in progress between the first client and the server
- d_2 : a request (d) is in progress between the second client and the server

Kripke Structure for the example

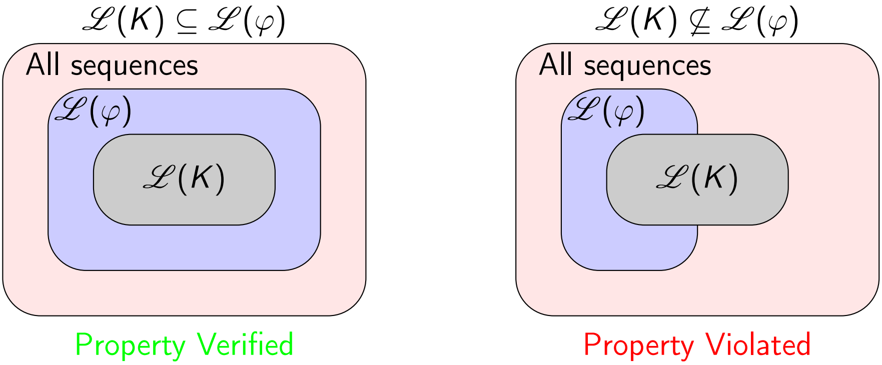


Automata approach for model checking

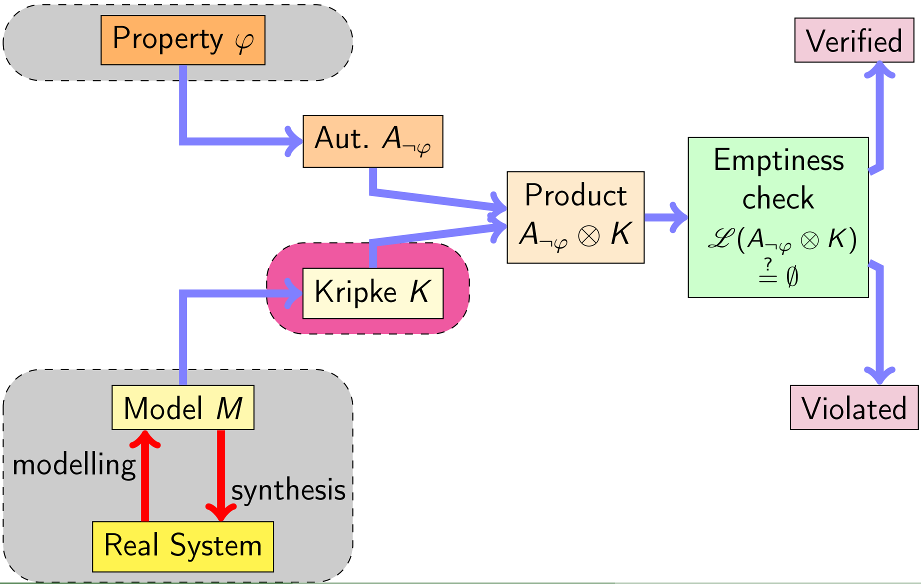


Why to check $\mathscr{L}(A_{\neg\varphi} \otimes K) \stackrel{?}{=} \emptyset$?

We want to check $\mathscr{L}(K) \subseteq \mathscr{L}(\varphi)$, which is equivalent to check $\mathscr{L}(K) \cap \overline{\mathscr{L}(\varphi)} \stackrel{?}{=} \emptyset$, which is equivalent to check $\mathscr{L}(A_{\neg\varphi} \otimes K) \stackrel{?}{=} \emptyset$



Automata approach for model checking



Express Property Automaton

How to express?

If client 1 send a request, he will necessarily receive a response

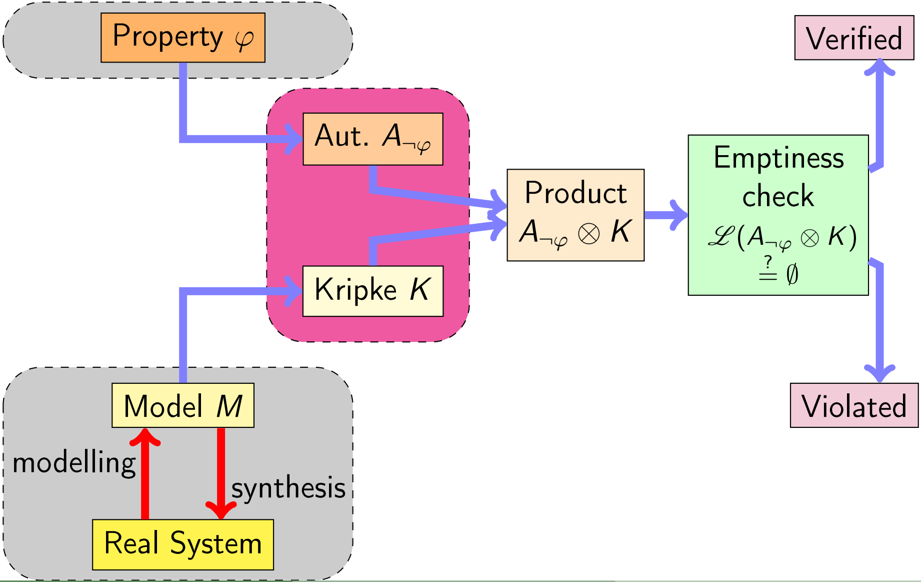
Express Property Automaton

How to express ?

If client 1 send a request, he will necessarily receive a response

$$'(G(d_1 \text{ -> } F\ r_1))'$$

Automata approach for model checking



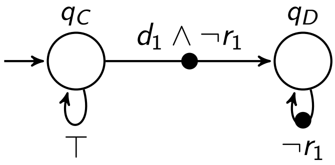
Express Property Automaton

How to express ?

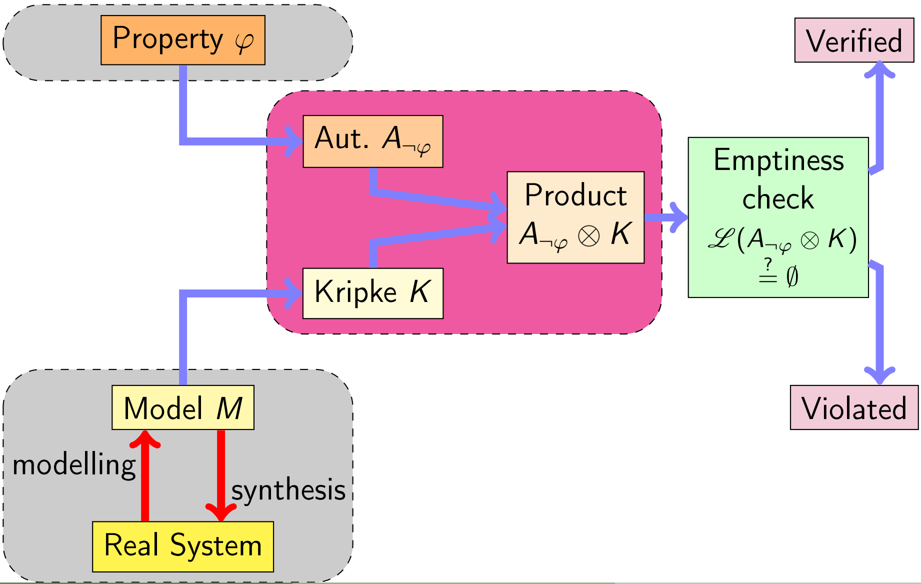
If client 1 send a request, he will necessarily receive a response

'(G(d_1 -> F r_1))'

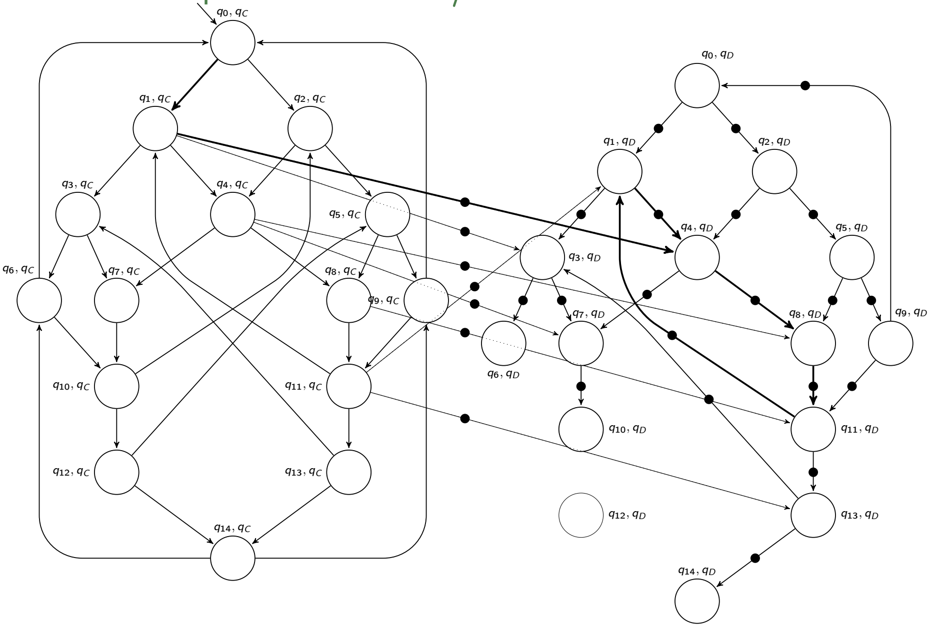
We can translate '!(G(d_1 -> F r_1))' into an automaton :



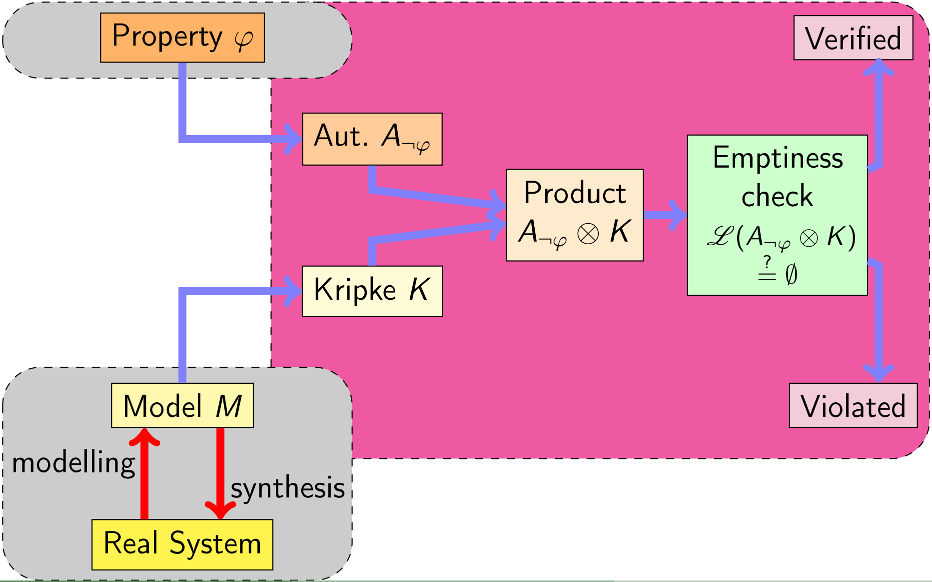
Automata approach for model checking



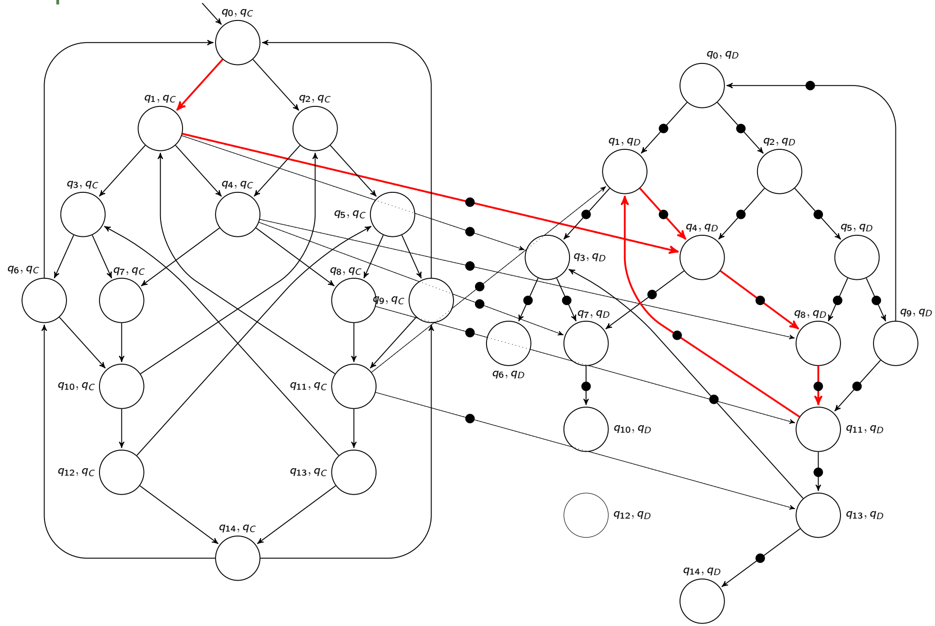
Product Kripke structure / Automaton



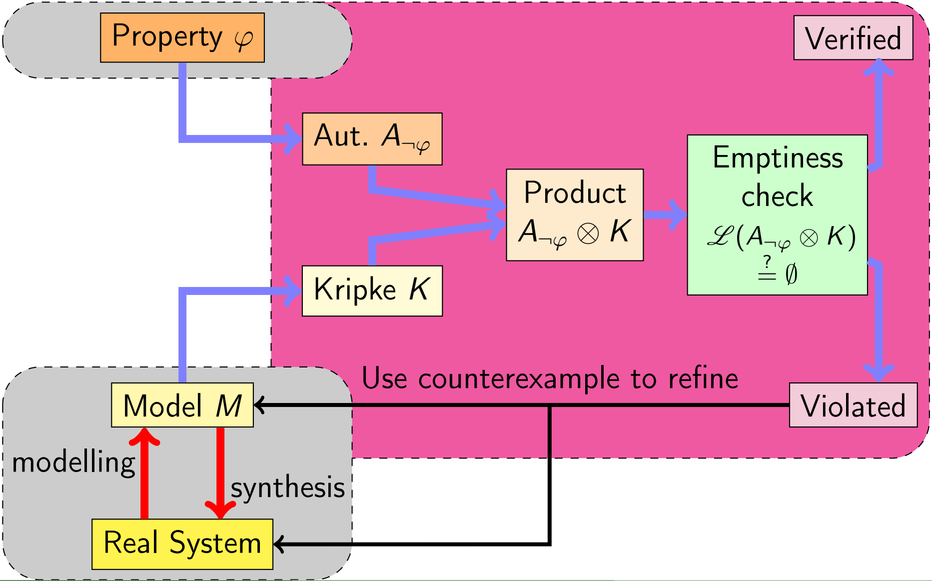
Automata approach for model checking



Emptiness check



Automata approach for model checking



Sum up

- From a model, we can build the kripke structure if :
  - we can extract the initial state
  - we can compute the successors of a given state
- Divine2.4 tool (patch by LTSmin) build such a Kripke structure
  - from the DVE language
  - spot can read kripke structures generated by Divine2.4
  - BNF for DVE can be found (page 8 – 9) at `https://is.muni.cz/www/208047/meandve.pdf`