kPWorkbench: A software framework for Kernel P systems

Laurențiu Marian Mierlă

13th BWMC

Outline

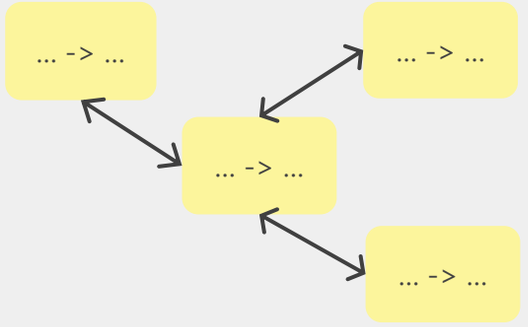
1. Kernel P systems
2. kPWorkbench
3. Case studies
4. Demo
5. Q&A

Outline

1. Kernel P systems
2. kPWorkbench
3. Case studies
4. Demo
5. Q&A

Kernel P systems

1. Membrane structure



2. Guarded rule execution

r1: ... -> ... $\{\leq a^{10} = b^2\}$
r2: ... -> ... $\{\geq p^{10} = q\}$
r3: ... -> ... $\{\neq a^2 \leq b\}$

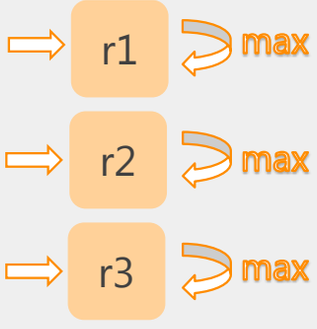
3. Rewriting and communication rules

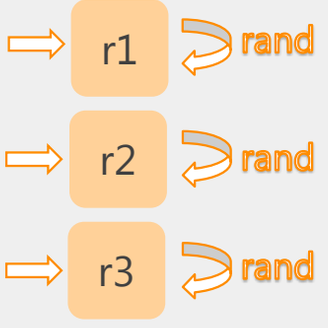
r1: $a^{10} \rightarrow b,\ c^2,\ (d, t_1)$
r2: $pq^2 \rightarrow (k, t_1), (a^5, t_2)$
r3: $x^{10} \rightarrow \lambda$

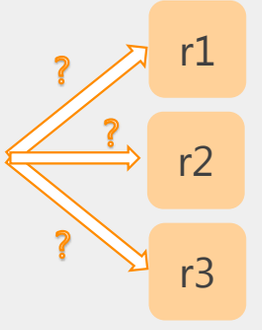
4. Structural rules

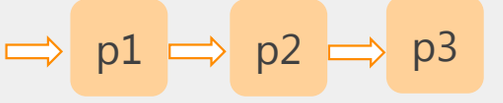
r1: $[\,]_{t_1} \rightarrow [\,]_{t_1}[\,]_{t_2}[\,]_{t_3}$
r2: $[a, b]_{t_1} \rightarrow [a]_{t_1}[b]_{t_2}$
r3: $[\,]_{t_1} \rightarrow \lambda$

4. Execution strategies









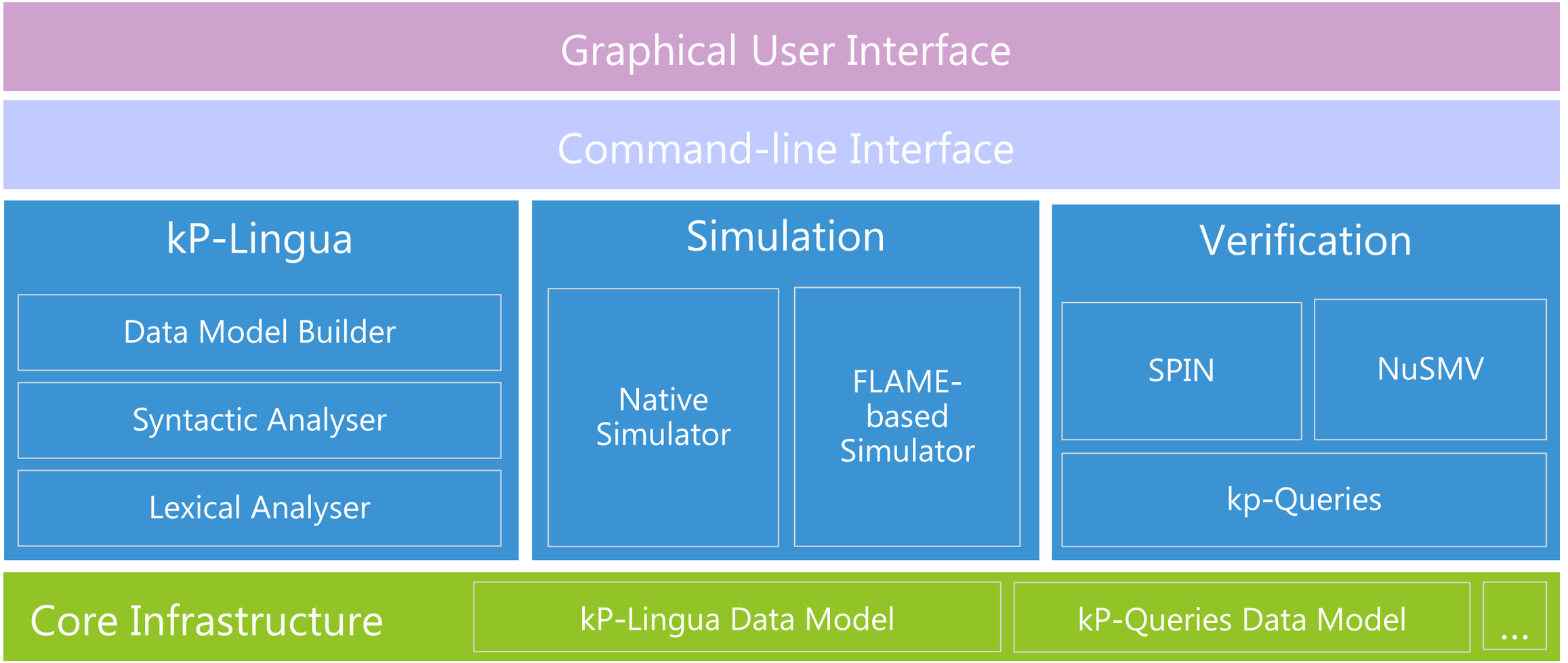
Outline

1. Kernel P systems
2. kPWorkbench
3. Case studies
4. Demo
5. Q&A

What is kPWorkbench?

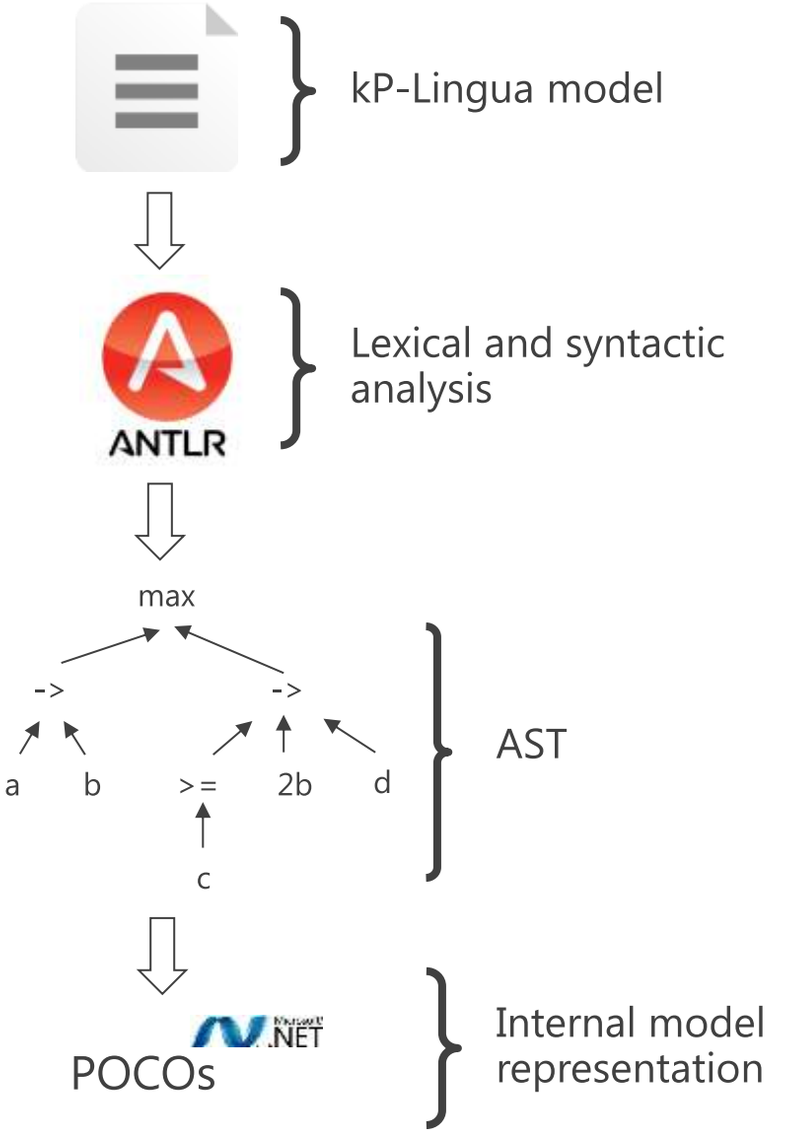
- Integrated software suite aimed to provide tool support for Kernel P systems
- Provides tool for modelling, simulating and verifying Kernel P systems
- Written in C#, using the .NET platform
- CLI for easier integration with other tools or scripts
- GUI for easier end-user interaction in modelling, simulation and verification

KPWorkbench tool stack



Modeling in kPWorkbench

- Intuitive and coherent modelling language – kP-Lingua
- Uses ANTLR as a parser generator, providing the EBNF grammar of the kP-Lingua DSL
- AST traversal implemented using the Visitor design pattern
- An internal model representation is generated and passed to the different modules for analysis



kP-Lingua constructs

```
type C1 {
    2a, 3b -> c .
    arbitrary {
        >= 2c & > 2b : b, c -> a .
    }
    choice {
        b -> 2b .
        < 3b : b -> 3b .
    }
    max {
        a -> a, a(C2), {a, 2b}(C3) .
    }
    2c -> - (C2) .
    2b -> \- (C2) .
    = 5a : a -> [3a, 3b](C1) [3b](C2) .
}
```

- Compartment type definitions
- Rewriting and communication rules
- Rule guards
- Membrane division
- Link creation and destruction
- Sequential, arbitrary, choice and maximal parallelism execution strategies

```
m1 {2x, b} (C1) .
m2 {x} (C2) .
```

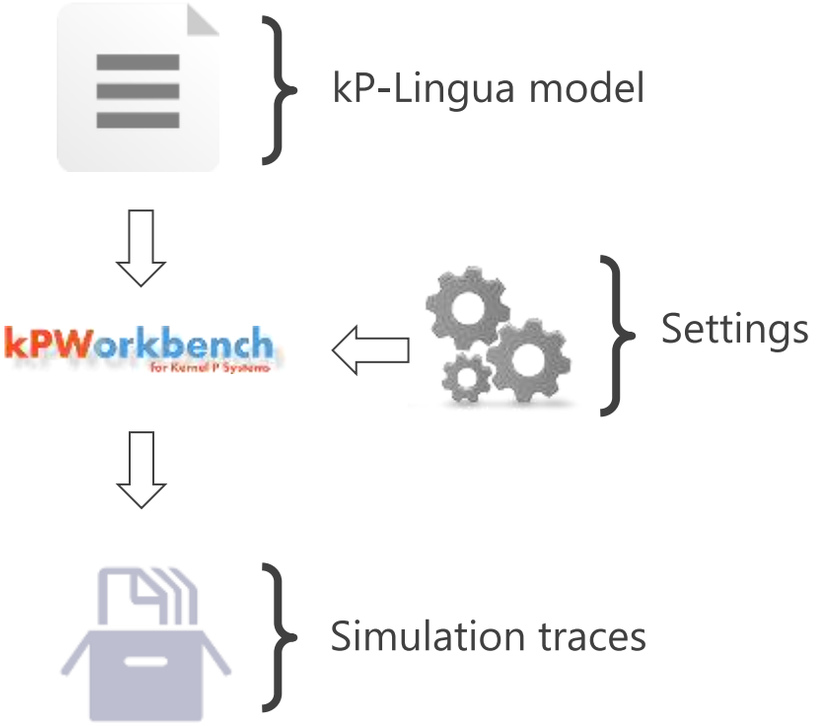
- Membrane instantiations

```
m1 - m2 .
```

- Link between compartments

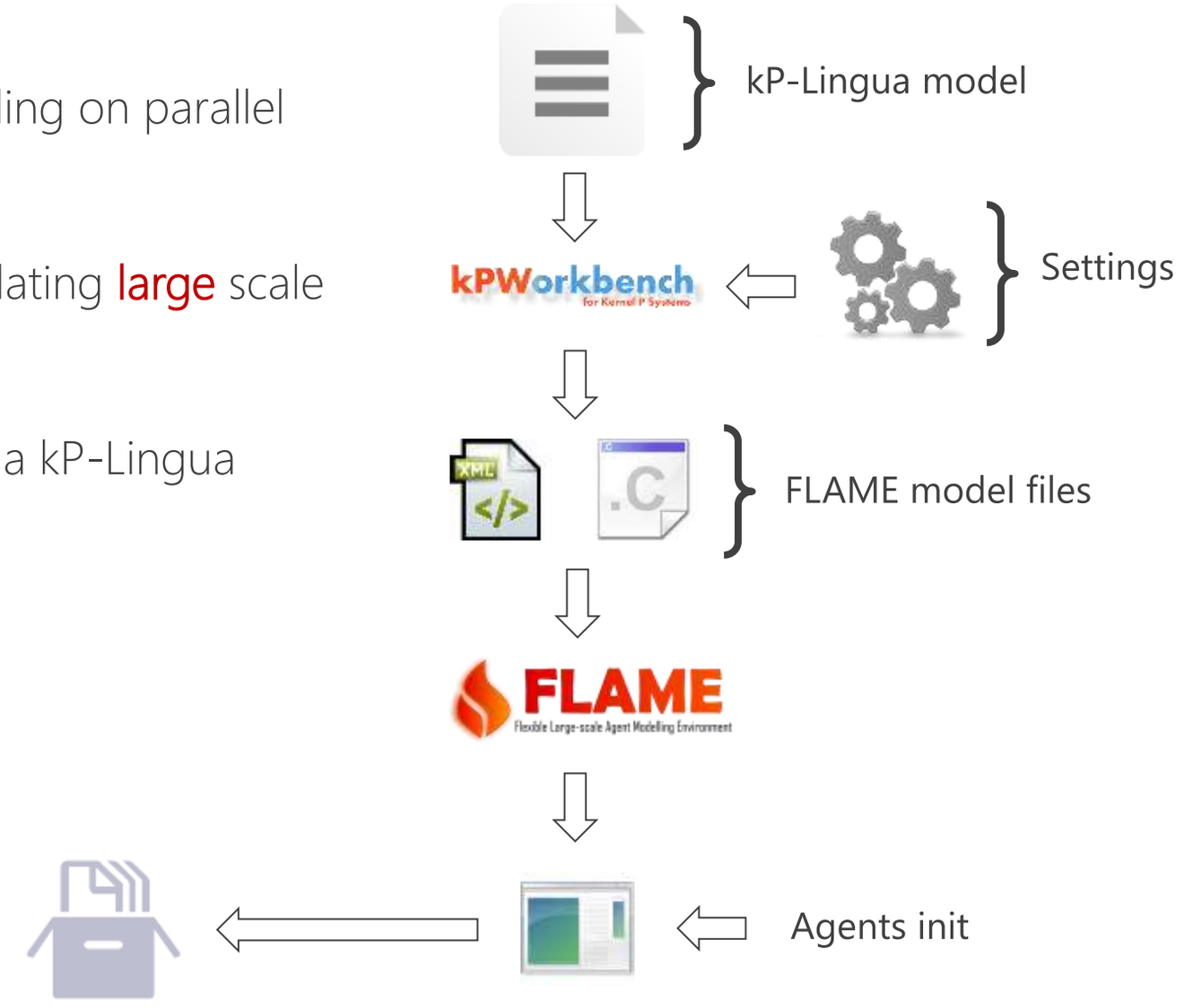
Simulation

- Simulation traces permit explores the dynamics of the system and its evolution over time
- Two simulation approaches: the Native Simulator and a FLAME-based Simulator
- Native Simulator - implemented is C#, most suitable for small to medium-size models



FLAME-based simulator

- FLAME - A platform for agent-based modeling on parallel architectures
- Big scalability degree and efficiency in simulating large scale models
- Automated FLAME model generation from a kP-Lingua specification

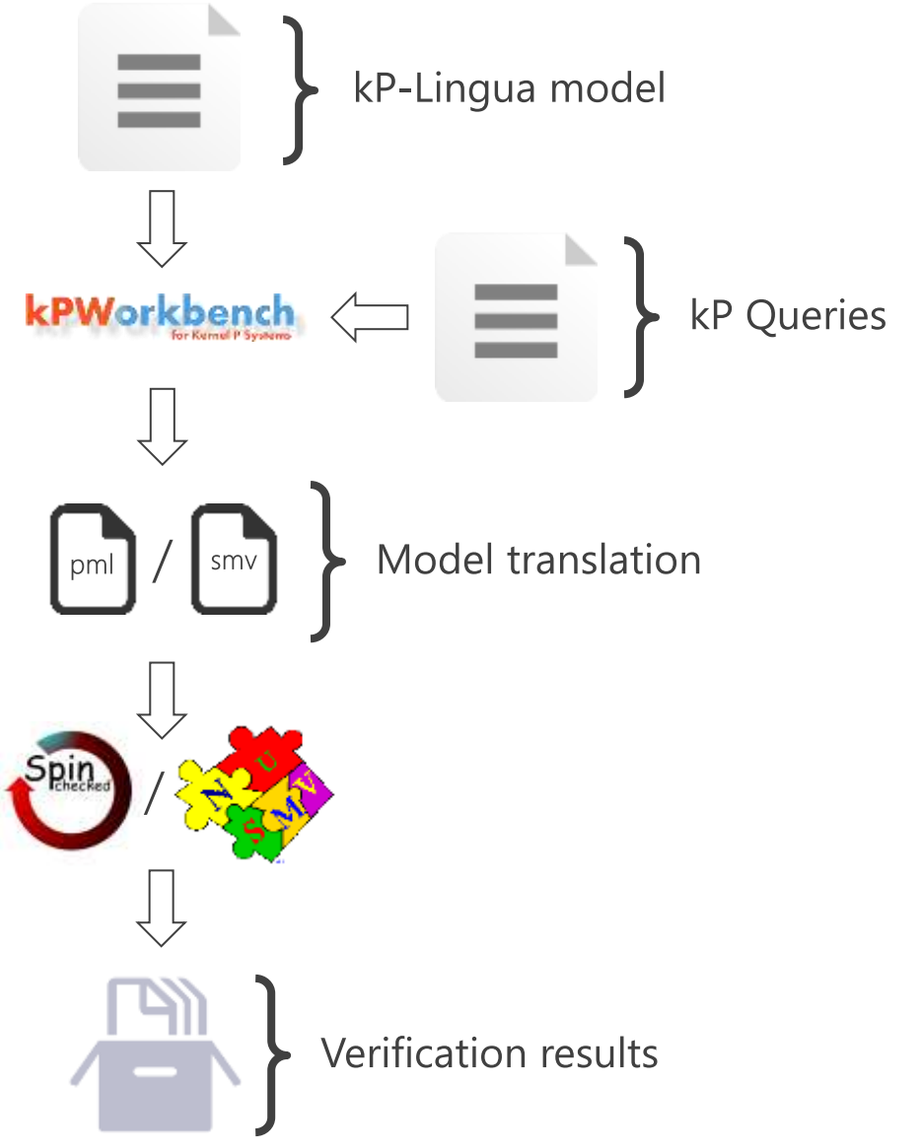


kP-lingua to FLAME mapping

kP-Lingua constructs	FLAME constructs
Compartment type	Agent definition, Communicating Stream X-Machine
kP system membrane	Agent instance
Membrane multi-set	Agent data
Membrane rules	Agent data
Execution strategy	States and transitions into the X-Machine, associated C functions
Communication rule	Message passing
P system steps	Synchronisation with message passing

Verification

- Integrates two state of the art model checkers – SPIN and NuSMV
- Automated translations from kP-Lingua models to the target representations: Promela and SMV
- kP-Queries – property specification language based on natural language statements
- kP-Queries follow a certain set of property patterns
- kP-Queries permits the specification of the target logic (LTL and CTL) for each property pattern



kP-lingua to SMV mapping

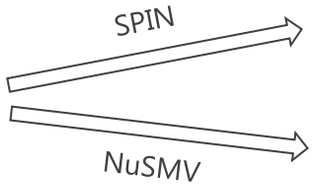
kP-Lingua constructs	SMV constructs
Compartment types	Module definitions
kP system membranes	Module instances
Membrane multi-sets	Module variables
Rewriting rules	*next* statements, transition relation of the FSM
Guards	*case* branches associated to the *next* statements
Execution strategies	Certain conditions associated to *case* branches
Communication rules	*next* statements into the *main* module
P system steps	Implicitly handled

kP-lingua to Promela mapping

kP-Lingua constructs	Promela constructs
Compartment types	Data type definitions
kP system membranes	Instances of data type definitions
Membrane multi-sets	Values of an integer array, indexed by object IDs
Rewriting and communication rules	Subtraction/addition instruction sets
Execution strategies	Multi-branch do – od and if – fi non-deterministic statements
P system steps	A dedicated scheduler process

kP-Queries

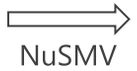
```
ltl: eventually m1.a > 0;
```




```
ltl prop1 { <> (c[0].x[a_] > 0 && state == step_complete) }
```

```
LTLSPEC F (m1.a > 0)
```

```
ctl: m2.a = 1 followed-by m3.a = 1;
```




```
SPEC AG (m2.a = 1 -> EF m3.a = 1)
```

Pattern	kP-Query	LTL	CTL
Next	next p	X *p*	EX *p*
Existence	eventually p	F *p*	EF *p*
Absence	never p	¬(F *p*)	¬(EF *p*)
Universality	always p	G *p*	AG *p*
Recurrence	infinitely-often p	G F *p*	AG EF *p*
Steady-state	steady-state p	F G *p*	AF AG *p*
Until	p until q	*p* U *q*	A (*p* U *q*)
Response	p followed-by q	G (*p* → F *q*)	AG (*p* → EF *q*)
Precedence	p preceded-by q	¬(¬*p* U (¬*p* ∧ *q*))	¬(E (¬*p* U (¬*p* ∧ *q*)))

Outline

1. Kernel P systems
2. kPWorkbench
3. Case studies
4. Demo
5. Q&A

Case studies

- Successfully used the methodologies in modelling, analysis and verification of well-known and unconventional case studies
- Illustrated also on case studies from systems and synthetic biology
- Studied phenomena in genetic regulatory networks, molecular interactions – non-deterministic models and qualitative analysis
- New case studies: Square numbers generation, Broadcasting with acknowledgement

Square numbers generation

```
type main {
    max {
        = t: a -> {} .
        < t: a -> a, 2b, s .
        < t: a -> a, s, t .
        < t: b -> b, s .
    }
}

m {a} (main) .
```

```
/* LTL Properties */
ltl: always m.t <= 1;
ltl: steady-state (m.a = 0 implies m.t = 1);
ltl: never m.s = 15;


/* CTL Properties */
ctl: eventually m.a = 0;
ctl: eventually m.t = 1;
ctl: m.a = 0 preceded-by m.t = 1;
ctl: m.a > 0 followed-by m.a = 0;
ctl: m.t = 1 followed-by m.a = 0;
ctl: always m.t <= 1;
ctl: never m.s = 15;
```

Broadcasting with acknowledgement

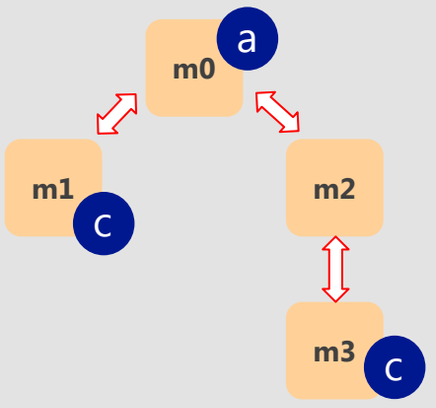
```
type L0 {
    max {
        a -> b, a (L1), a (L2) .
    }
}

type L1 {
    max {
        a, c -> c (L0) .
    }
}

type L2 {
    max {
        a -> b, a (L3) .
        b, c -> c (L0) .
    }
}
```

```
type L3 {
    max {
        a, c -> c (L2) .
    }
}

m0 {a} (L0) .
m1 {c} (L1) - m0 .
m2 {} (L2) - m0 .
m3 {c} (L3) - m2 .
```




```
/* LTL Properties */
ltl: eventually (m1.a > 0 and m3.a > 0);
ltl: m2.a = 1 followed-by m3.a = 1;
ltl: never m0.a > 0 and m0.c > 0;

/* CTL Properties */
ctl: eventually m1.a > 0;
ctl: eventually m1.a > 0 and m3.a > 0;
ctl: m2.a = 1 followed-by m3.a = 1;
ctl: steady-state (m0.c = 2 implies m0.a = 0);
ctl: steady-state (m0.c = 2 implies
          (m1.c = 0 and m3.c = 0));
ctl: never m0.a > 0 and m0.c > 0;
```

Demo

Hands-on kPWorkbench

Q&A

KPWorkbench Platform Website

http://kpworkbench.org/