Intro to UPPAAL

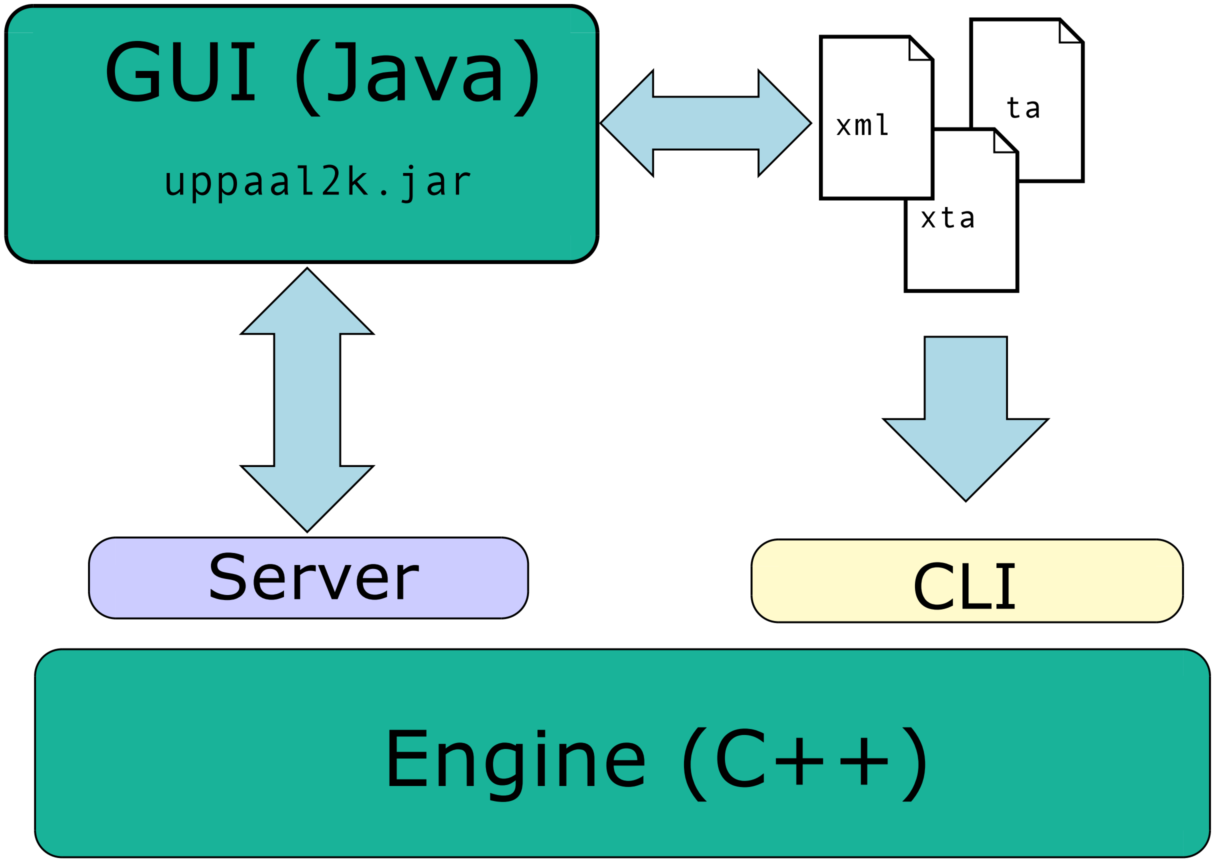
Gerd Behrmann
Kim Larsen

BRICS & Aalborg University

Plan of the Lecture

1. UPPAAL Architecture
2. UPPAAL Features
3. Train Gate Example
4. Demonstration of Uppaal

UPPAAL's Architecture



Short view of UPPAAL

Declarations

Clocks

```
clock x1, x2,...,xn;
```

Bounded Integer Variables

```
int[0,5] i1, i2,... in;
```

Constants

```
const delay 5, true 1, false 0;
```

Arrays

```
int x[4] := { 1, 4, 7, 2 }
```

Expressions

```
Expression
   ::= ID
   |   NAT
   |   Expression '[' Expression ']'
   |   '(' Expression ')'
   |   Expression '++' | '++' Expression
   |   Expression '--' | '--' Expression
   |   Expression AssignOp Expression
   |   UnaryOp Expression
   |   Expression BinOp Expression
   |   Expression '?' Expression ':' Expressio
   |   ID '.' ID
```

Operators

Unary

```
'-' | '!' | 'not'
```

Binary

```
'<'   | '<=' | '==' | '!=' | '>=' | '>'
'+'   | '-'  | '*'  | '/'  | '%'  | '&'
'|'   | '^'  | '<<' | '>>' | '&&' | '||'
'and' | 'or' | 'imply'
```

Assignment

```
':=' | '+=' | '-=' | '*=' | '/=' | '%='
'|=' | '&=' | '^=' | '<<=' | '>>='
```

Guards

Any expression satisfying the following conditions is a guard:

- It is side effect free, type correct and evaluates to a boolean.
- Only clock variables, integer variables and constants are referenced (or arrays of these types).
- Clocks and differences between clocks are only compared to integer expressions (no inequallity).
- Guards over clocks are essentially conjunctions (*i.e.* disjunctions are only allowed over integer conditions).

Assignments

Any expression satisfying the following conditions is an assignment:

- It has a side effect and is type correct.
- Only clock variables, integer variables and constants are referenced (or arrays of these types).
- Only integers are assigned to clocks.

Invariants

Any expression satisfying the following conditions is an invariant:

- It is side effect free and is type correct.
- Only clock variables, integer variavles and constants are referenced (or arrays of these types).
- It forms a conjunction of conditions on the form x < e or x <= e, where x is a clock reference and e evaluates to an integer.

Binary Synchronisation

Channels can be declared like:

```
chan a, b, c[3];
```

If a is channel, then:

- `a!` = Emission
- a? = Reception

Two edges in different processes can synchronise if one is emitting and the other is receiving on the same channel.

Broascast Synchronization

Broadcast channels can be de declared like:

```
broadcast chan a, b, c[2];
```

If a is a broadcast channel, then:

- `a!` = Emission of a broadcast
- a? = Reception of a broadcast

A set of edges in different processes can synchronise if one is emitting and the others are receiving on the same broadcast channel. A process can always emit on a broadcast channel.

Urgent Channels

No delay if the synchronising edges can be taken!

Restriction: No clock guard allowed on the edges.

Allowed: Guards on data-variables.

Urgent channels can be declared like:

```
urgent chan a,b,c[3];
```

Urgent Location

No delay in urgent location!

Time is freezed.

Remark: The use of the urgent location reduces the number of clocks in a model, and thus the complexity of the analysis.

Committed Location

No delay pass in committed location!

Next transition must involve an edge in one of the processes in a committed location.

Remark: The use of the committed location considerably reduces the state space.

Templates

- Templates can be instantiated to form processes.
- Templates are parameterised.

Example of parameter declaration of a template `A`:

```
(int v, const min, const max)
```

Example of instantiation:

```
P := A(i, 1, 5);
Q := A(j, 0, 4);
```

Example of system declaration:

```
system P, Q;
```

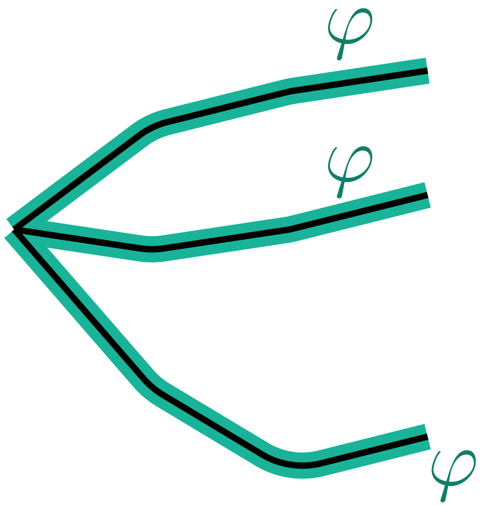
Syntax of Properties

```
A[] Expression
E<> Expression
A<> Expression
E[] Expression
Expression --> Expression
A[] not deadlock
```

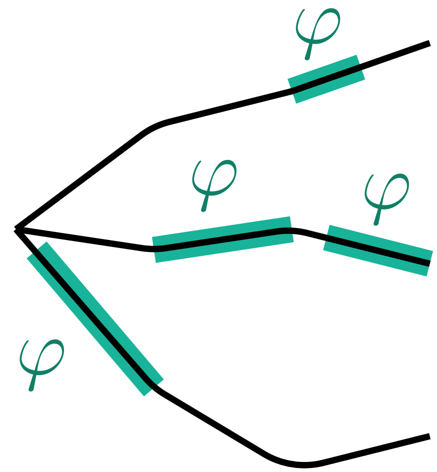
The expressions must be type safe, side effect free, and evaluate to a boolean. Only references to integers variables, constants, clocks, and locations are allowed (and arrays of these).

Operators A[ ] and A<>

A[]φ

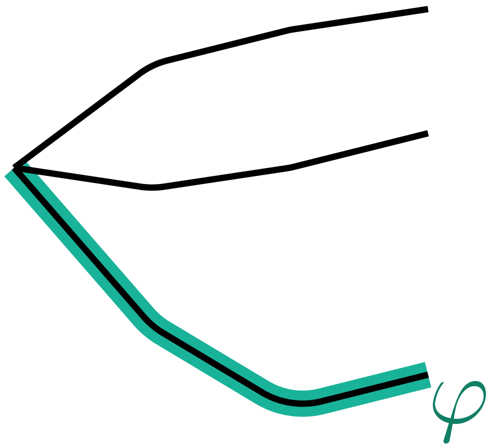


A<>φ

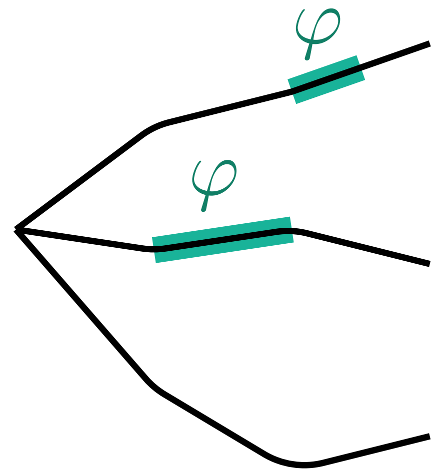


Operators E[ ] and E<>

$\texttt{E[]}\varphi$



$\texttt{E<>}\varphi$

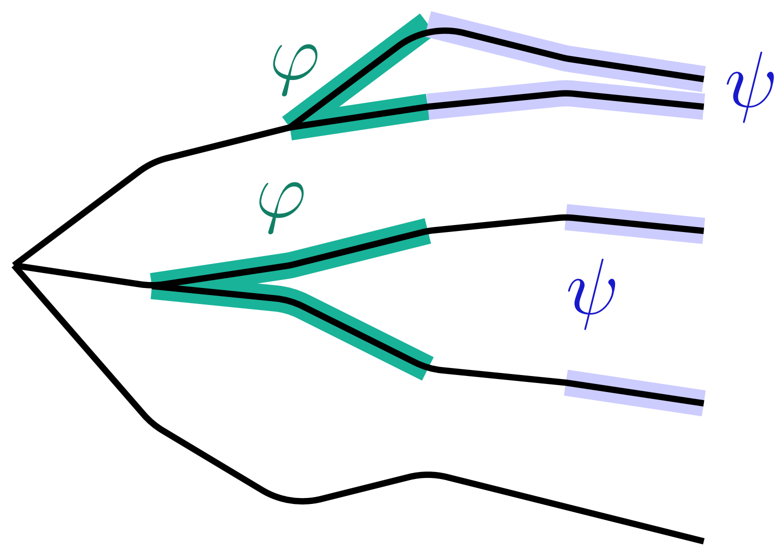


Remark

$\neg(\texttt{A[]}\varphi) = \texttt{E<>}(\neg\varphi)$ and $\neg(\texttt{E[]}\varphi) = \texttt{A<>}(\neg\varphi)$

Operator - - > (leads to)

$$\varphi \texttt{-->} \psi \overset{def}{\Longleftrightarrow} \texttt{A[]}(\varphi \Rightarrow \texttt{A<>}\psi)$$



State Property: `deadlock`

A deadlock is a state in which no action transition will ever be enabled again.

In other words $(l, u) \models \texttt{deadlock}$ iff:

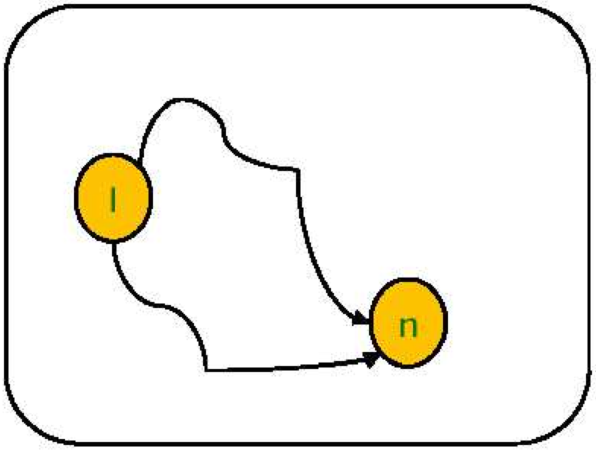
$$\forall d \geq 0, a \in \mathcal{A}ct : (l, u + d) \overset{a}{\not\rightarrow}$$

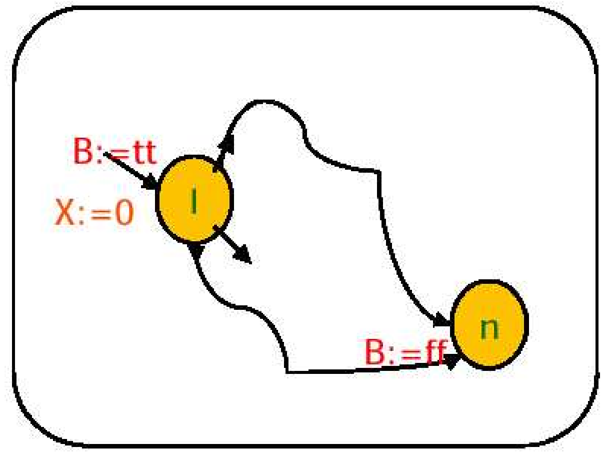
Checking for absence of deadlocks:

```
A[] not deadlock
```

Bounded Liveness: Decoration

$$AG(a \implies_{\leq t} b)$$





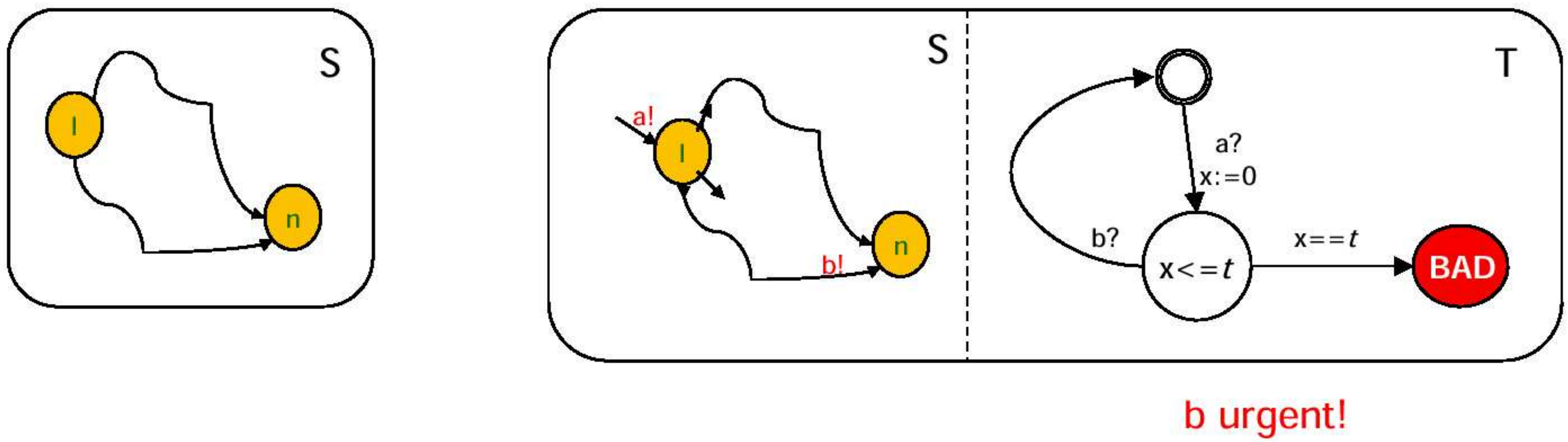
Leadsto:
Whenever l is reached
then n is reached with t

Decoration
new clock **X**
boolean **B**

```
A[] (B implies x <= t)
```

Test Automata

$$AG(a \implies_{\leq t} b)$$




```
A[] (not T.BAD)
```