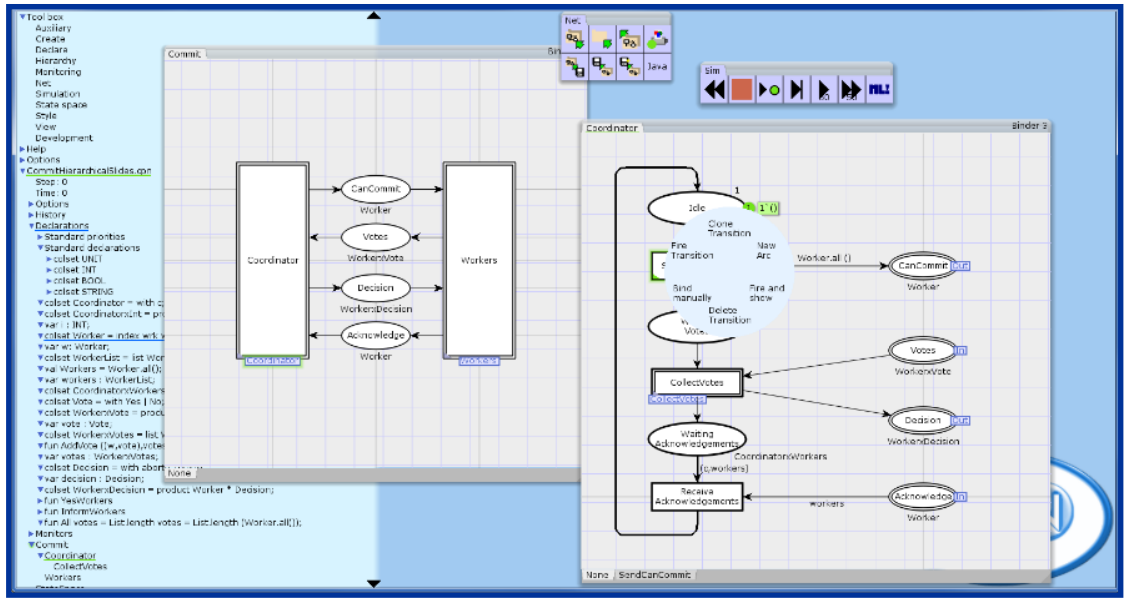
Theory-Tool | Part 3

Basic Concepts of Coloured Petri Nets: Syntax and Semantics

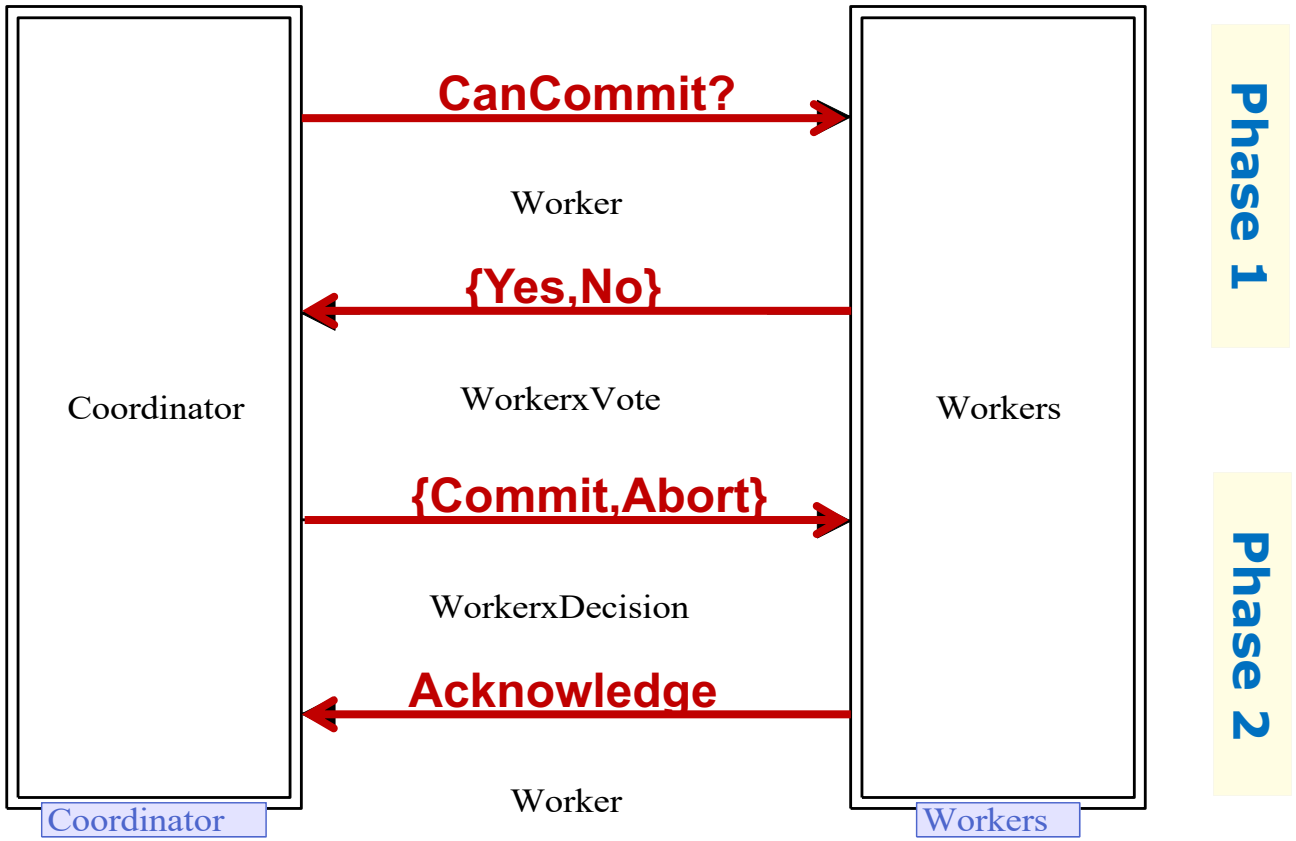


Lars M. Kristensen
Department of Computer Science, Electrical Engineering, and Mathematical Sciences
Western Norway University of Applied Sciences
Email: lmkr@hvl.no

Conceptual overview

- **Address the practical shortcomings of PT-nets**
- **Coloured Petri Nets (CPNs) =**
  PT-nets + Standard ML programming language
  - Places have a type (colour set) and tokens can carry data values
  - Transitions may have variables that can be bound to values (bindings)
  - Evaluation of arc expressions determine the tokens added/removed
  - Boolean guard expressions may be used as an extra enabling condition
- **Standard ML = functional programming**
  - Computation proceeds by evaluation of expressions
  - Static typing with the type of expressions being inferred
  - Functions are first-order values and can be polymorphic
  - Recursion and lists are used to express iteration

Two-phase Commit Transaction Protocol



Colour set definitions

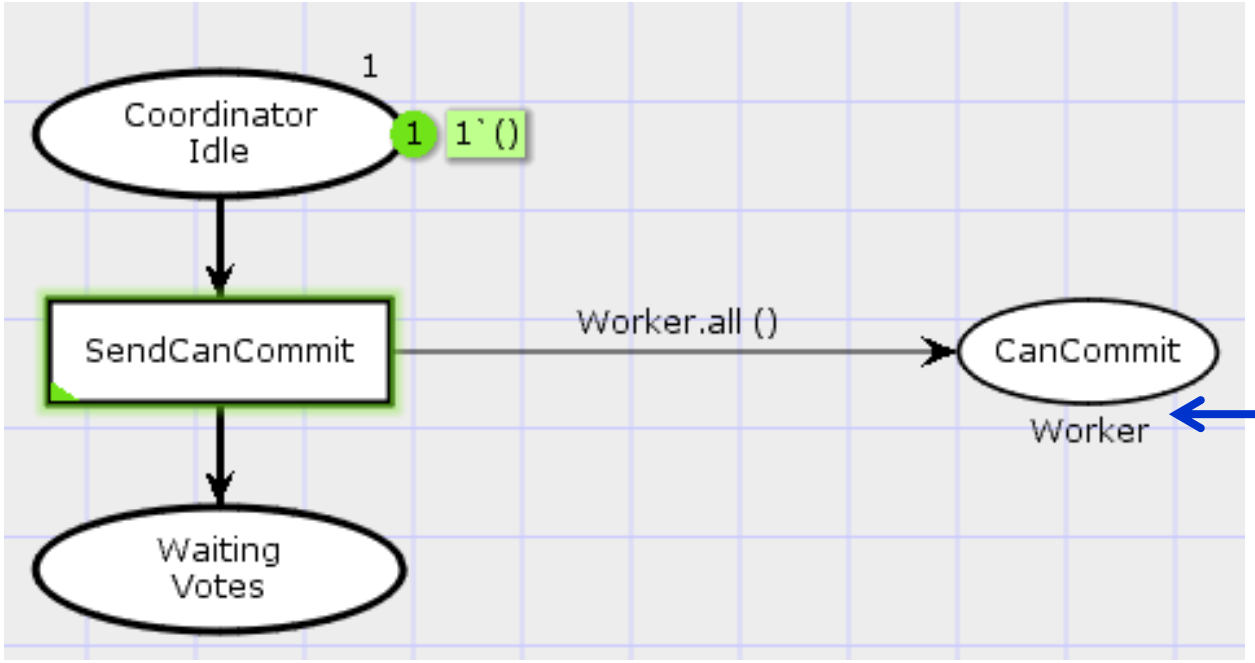
- **Determines the data types that can be used in the model**

Colour set definitions	Example values
`val W = 2;`	
`colset Worker = index wrk with 1..W;`	`wrk(1), wrk(2)`
`colset Vote = with Yes | No;`	`Yes, No`
`colset WorkerxVote = product Worker * Vote;`	`(wrk(1),Yes)`
`colset Decision = with Abort | Commit;`	`Abort, Commit`
`colset WorkerxDecision = product Worker * Decision;`	`(wrk(1),Commit)`

- **Additional colour set constructors for lists (`list`), records (`record`), and unions (`union`)**
- **Base data types: `UNIT`, `INT`, `STRING`, `BOOL`, `REAL`**

Coordinator – First phase

- **The colour set (type) of a place determine the kinds of tokens that may reside on the place**

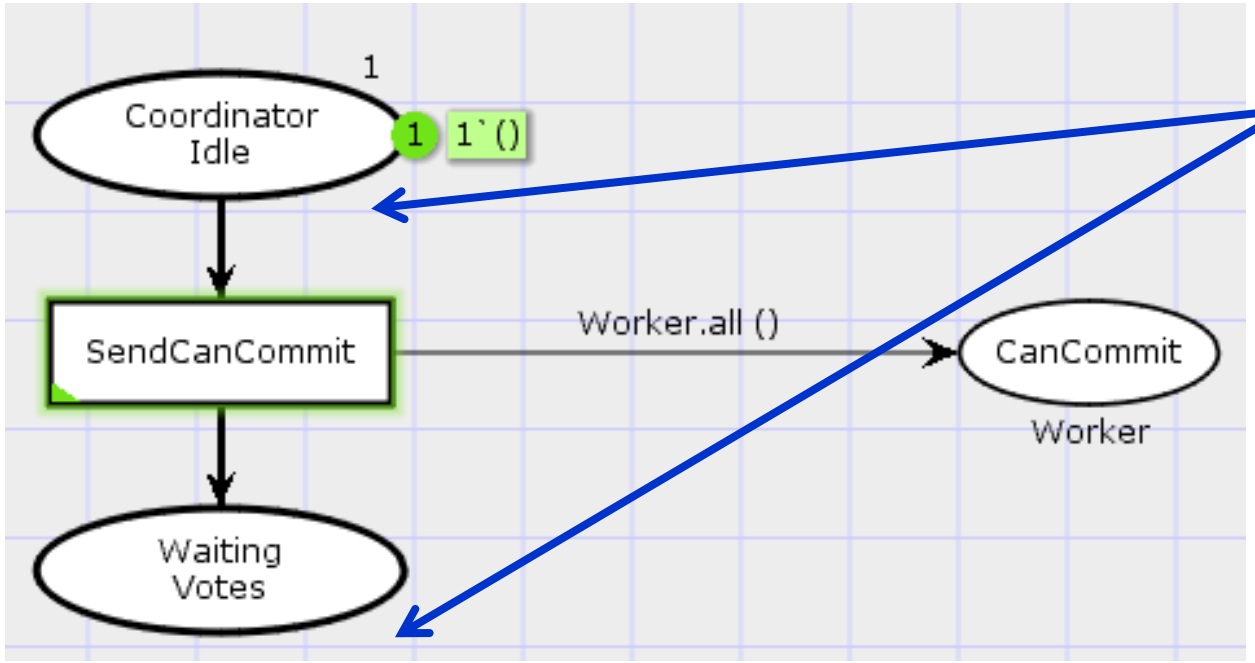


The colour set is by convention written below the place

Tokens on CanCommit can have the values `wrk(1)` and `wrk(2)`

The UNIT colour set

- **Absense of a colour set (implicitly) means that the place has the `UNIT` colour set**

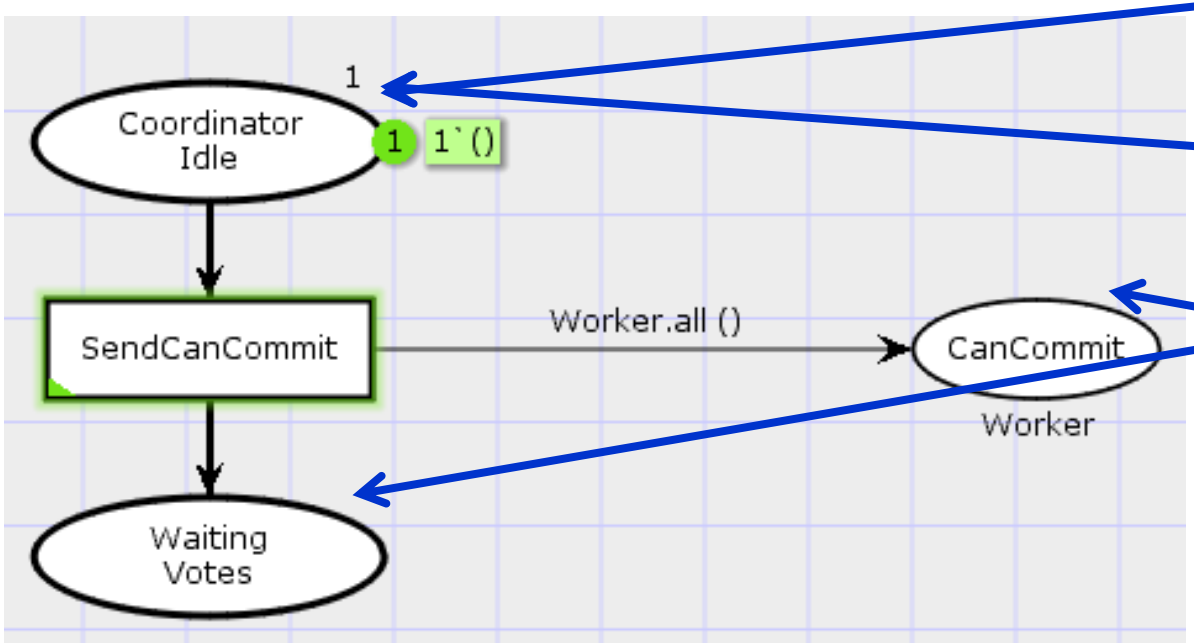


CoordinatorIdle and WaitingVotes have an implicit `UNIT` colour set

- **`UNIT` is a datatype containing the single value `()` named unit – used to represent the anonymous tokens of PT-nets**

Initial marking - Coordinator

- **The initial marking (state) is obtained by evaluating the initial marking expressions**



The initial marking is by convention written above the place

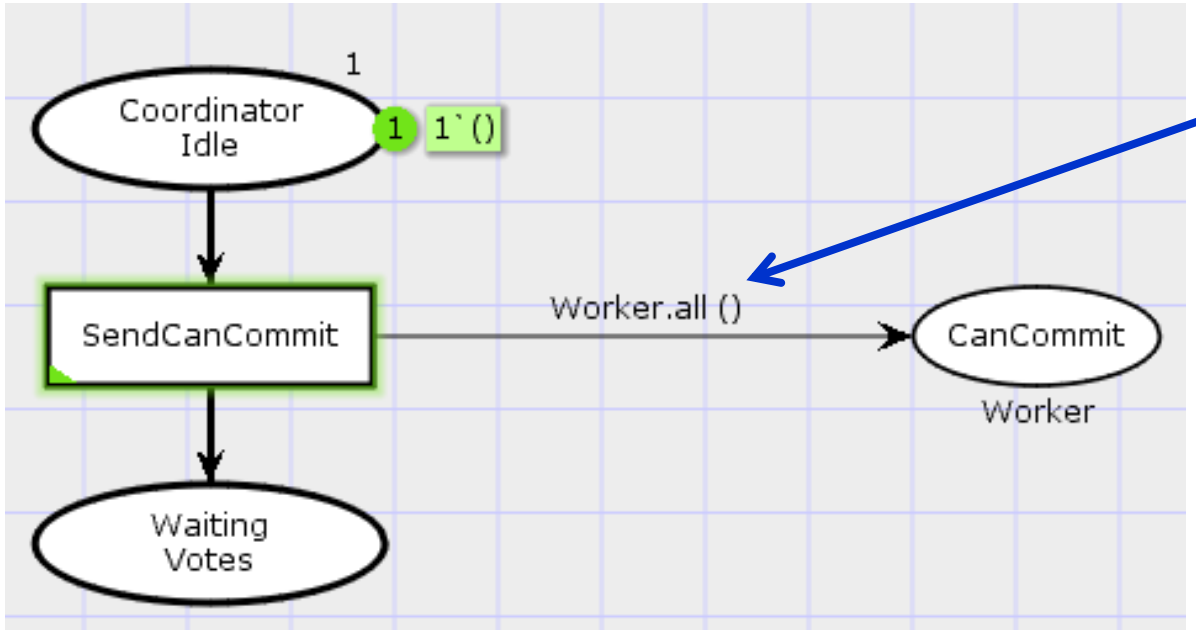
Initially one unit-token on CoordinatorIdle

Absense of an initial marking expression means no tokens

- **For a place with colour set `UNIT` the initial marking expression specifies the number of unit-tokens**

Arc expressions

- **Determine the tokens that are removed/added from/to places when transitions occur**



Expression evaluating to all values in the Worker colour set

```
Worker.all ()
```

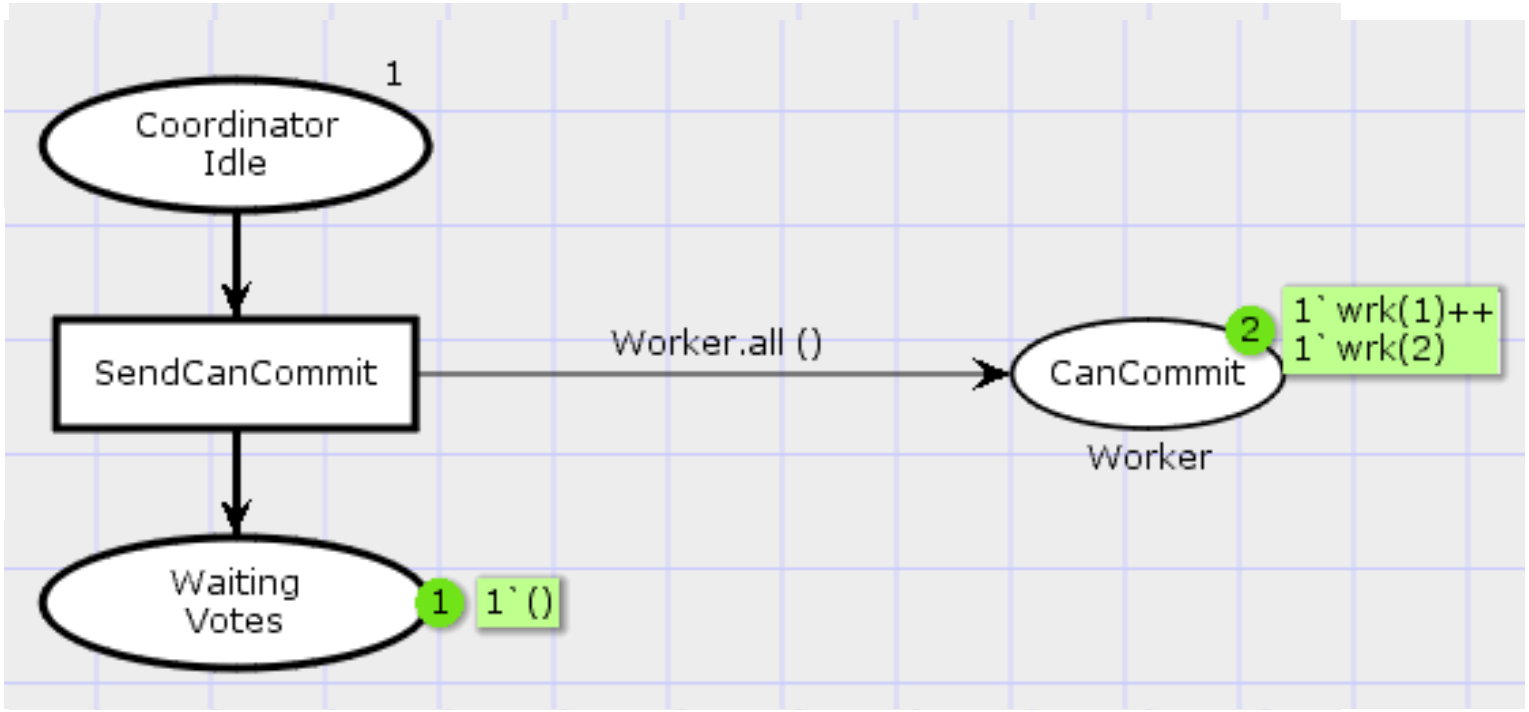
evaluate

```
[ wrk(1), wrk(2) ]
```

- **The type of an arc expression must match the colour set of the place connected to the arc**

Evaluation of expressions

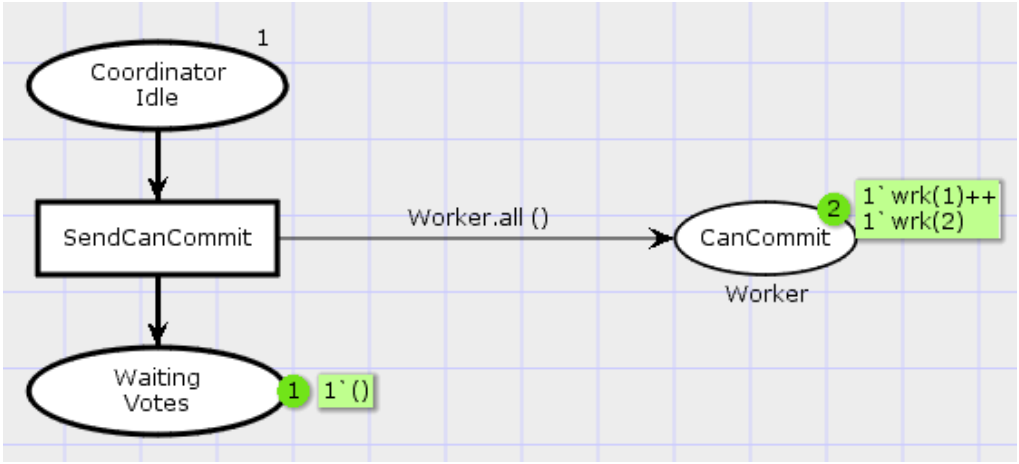
- **The tokens added and removed are determined by evaluating arc expressions**

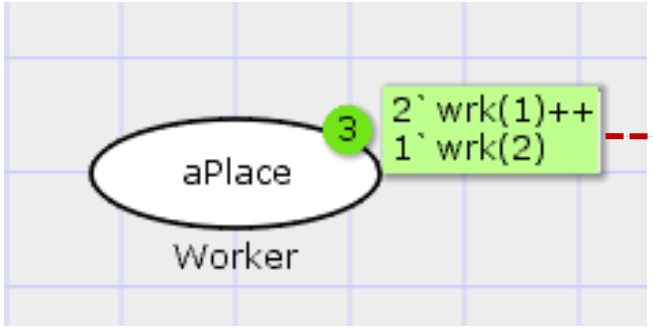


- **Absence of an arc expression (implicitly) means a single ()-token (a unit-token)**

Markings and multi-sets

- **Each place may hold a multi-set of tokens over the colour set of the place**





Multi-set notation

coefficient («of»)

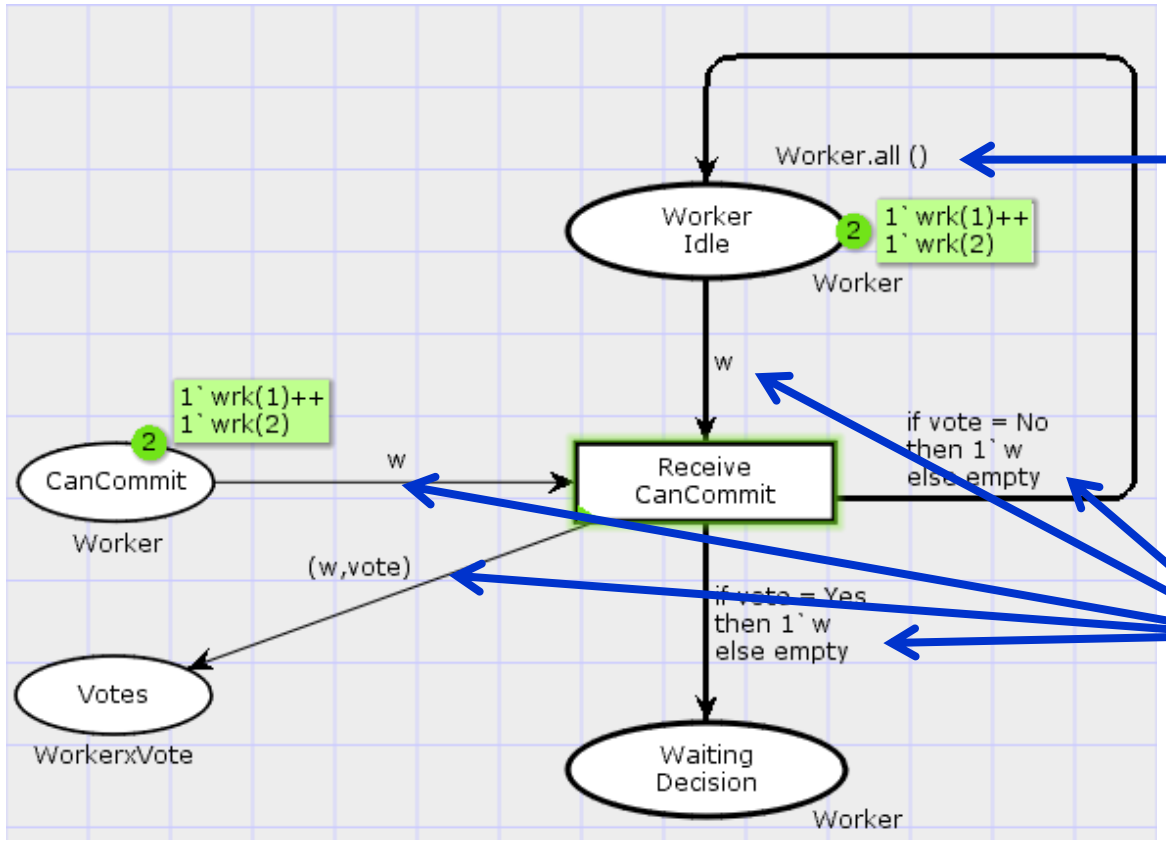
token colour (value)

```
2`wrk(1) ++
1`wrk(2)
```

union («and»)

Workers – First Phase

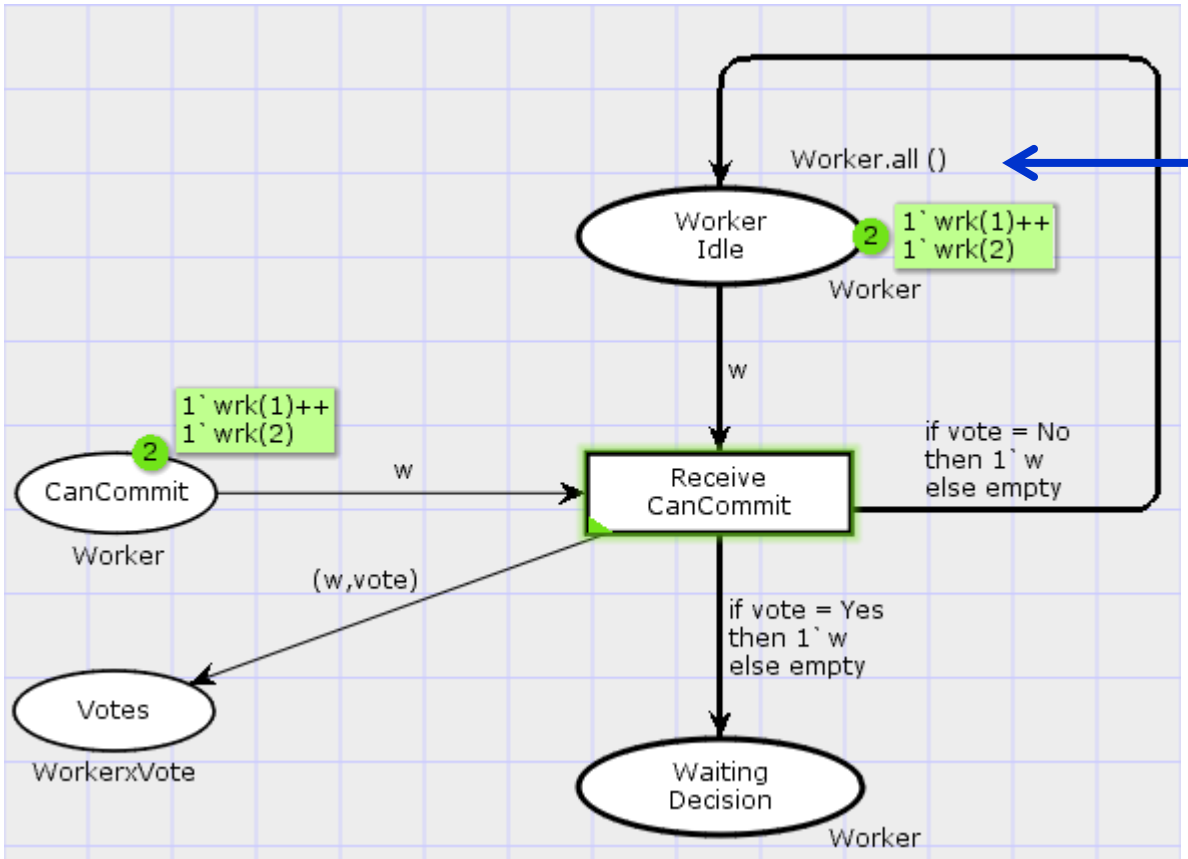
- **Consists of receiving a CanCommit message and sending a decision (Yes/No) to the coordinator**



Initial marking expression

Arc expressions with free variables (`w` and `vote`)

Initial marking - Workers



The two workers are initially Idle

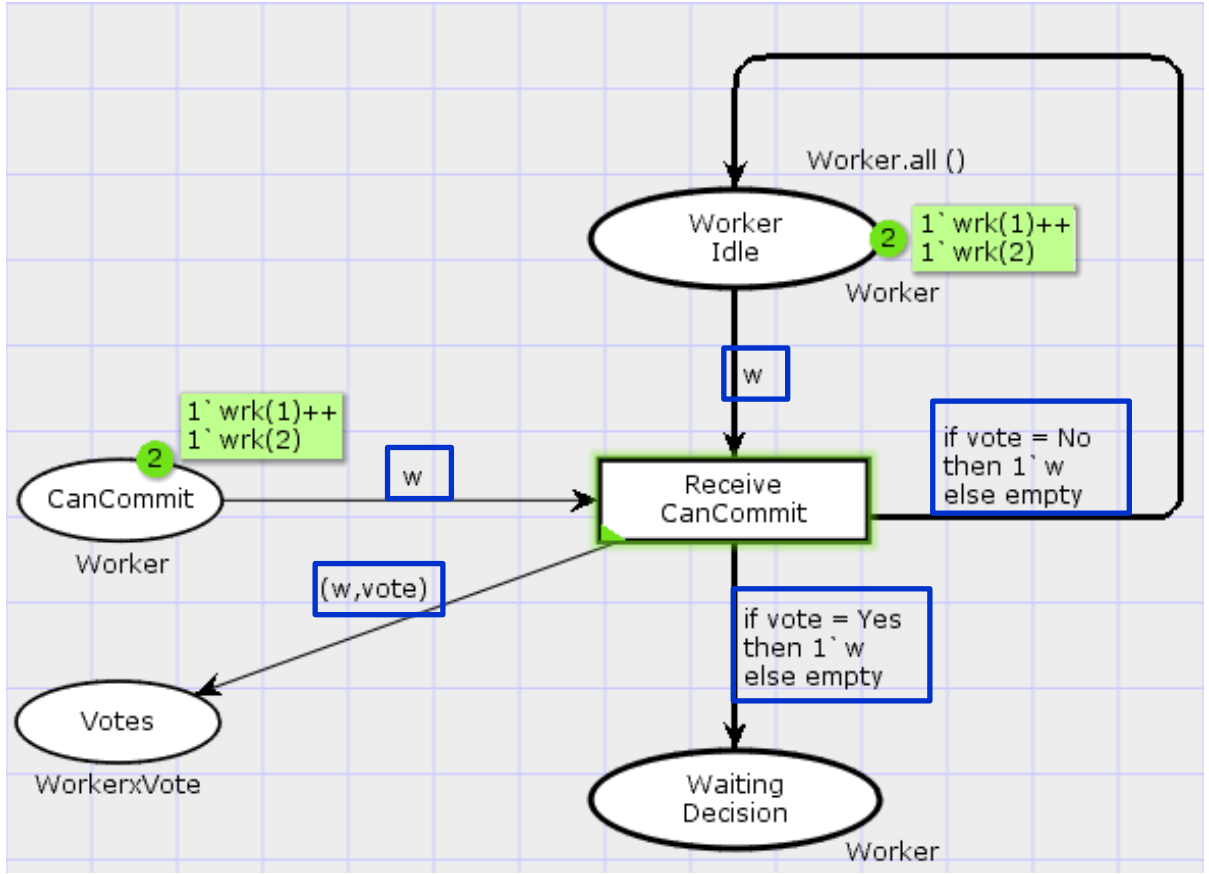
```
Worker.all ()
```

evaluate

```
[ wrk(1), wrk(2) ]
```

Transition variables

- **The arc expressions on the arcs of a transition may contain free variables**



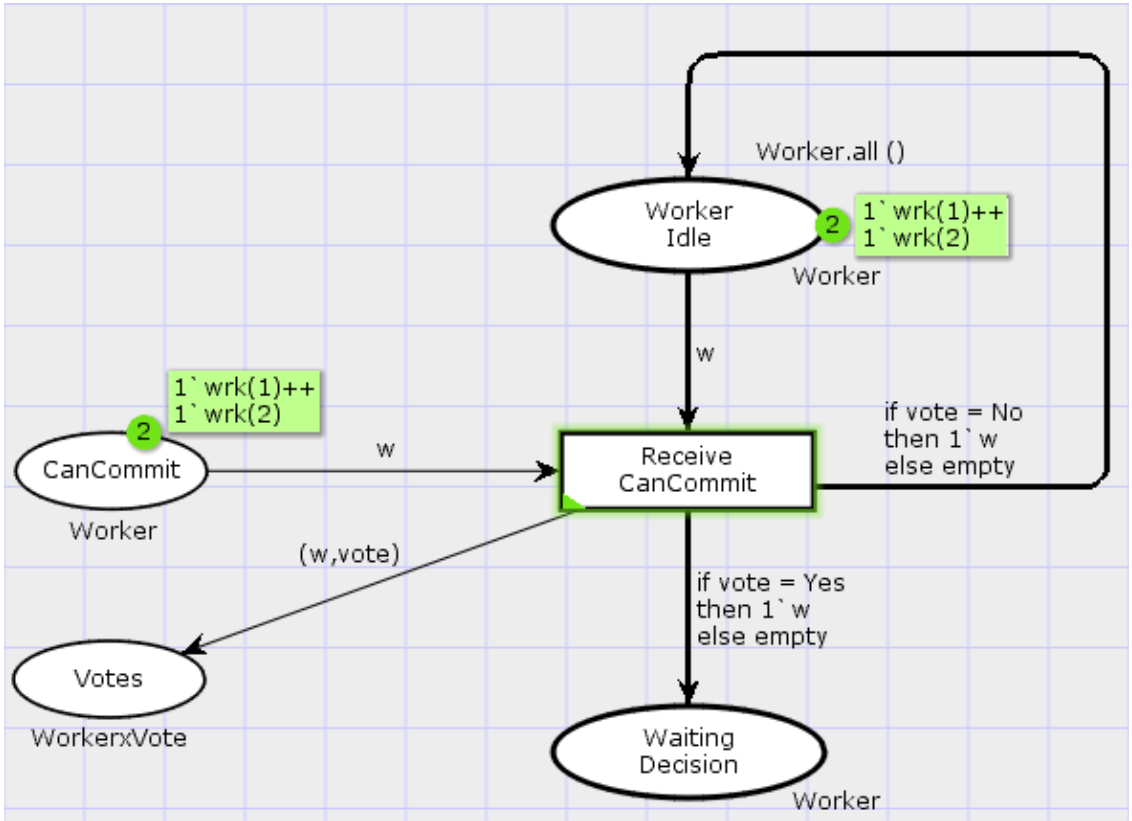
Variable declarations

```
val W = 2;
colset Worker =
        index wrk with 1..W;
var w : Worker;

colset Vote = with Yes | No;
var vote : Vote;
```

Arc expressions with free variables `vote` and `w`

Binding of transition variables

- The transition **ReceiveCanCommit** has two free variables: **vote** and **w**



- **Variables must be bound to values for a transition to be enabled and occur**
- **Similar to formal and actual parameters known from programming**
- **The association of values to variables is called a transition binding**
- **The bindings correspond to the possible enabling and occurrence modes of the transition**
- **Not all possible bindings will in general be enabled in a given marking**

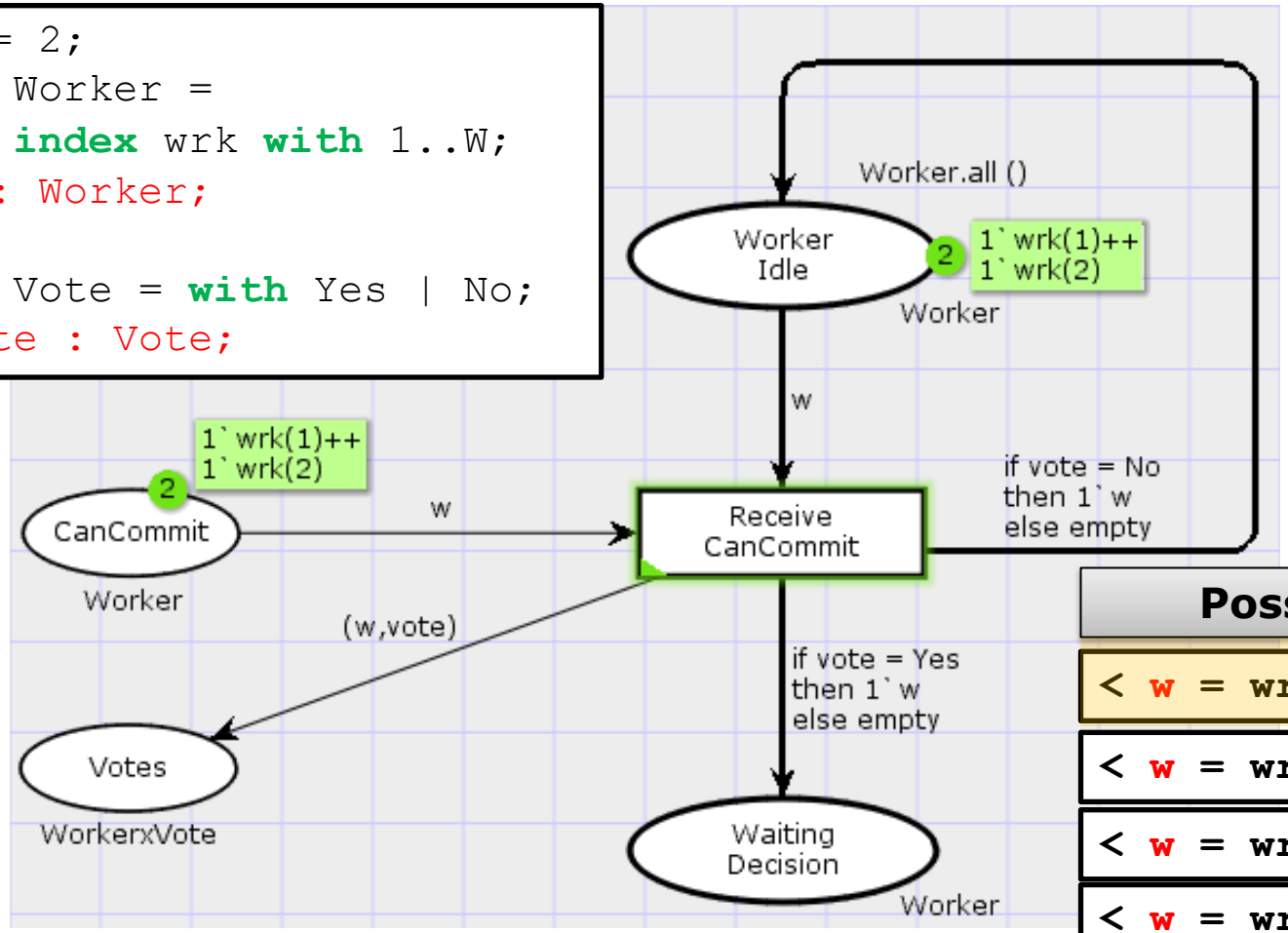
CPN Tools demo

`lecture3-cpns-simulation.cpn`

- **Simulation of CPN models**
  - Interactive simulation with binding selection
  - Returning to the initial marking

Transition bindings

```
val W = 2;
colset Worker =
        index wrk with 1..W;
var w : Worker;

colset Vote = with Yes | No;
var vote : Vote;
```



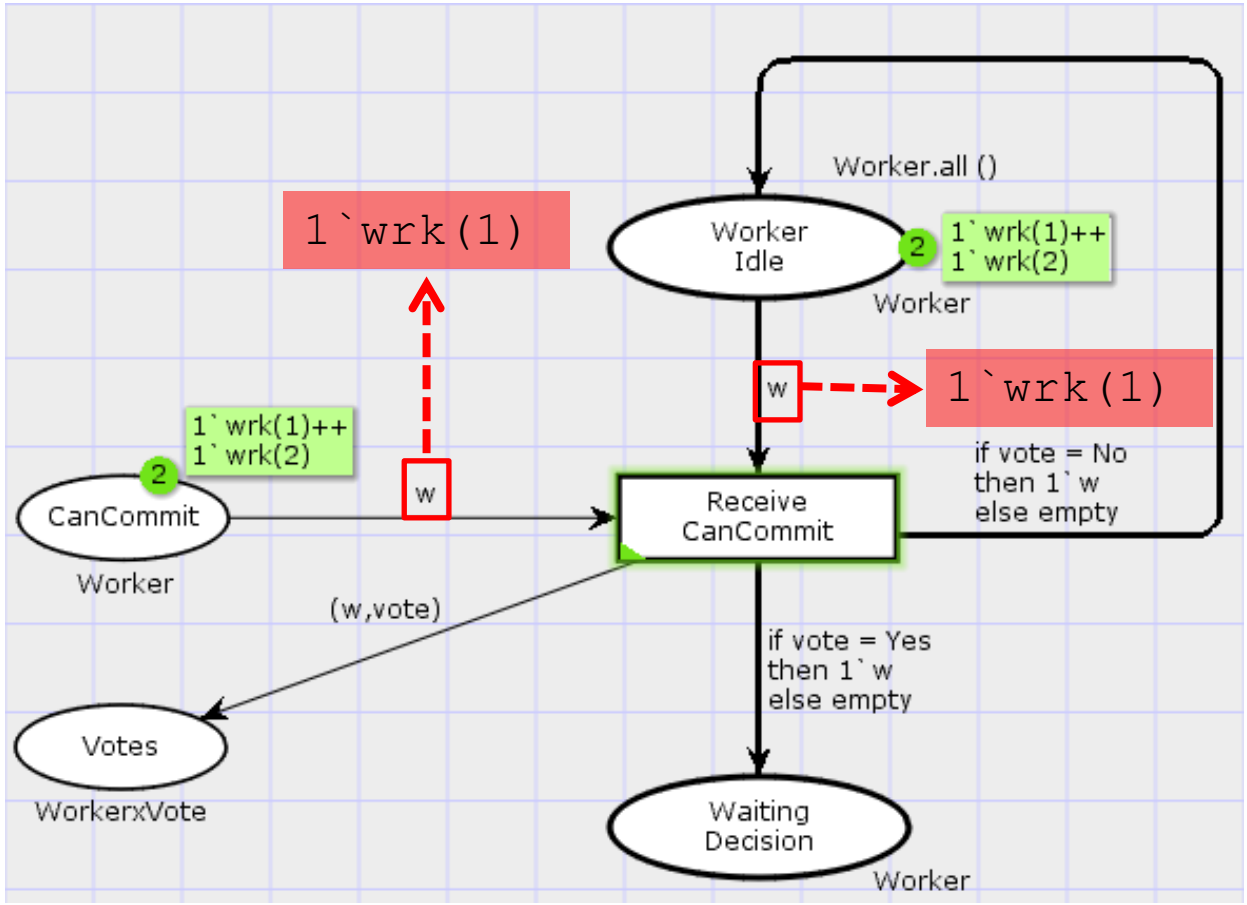
Possible bindings ?

- `< w = wrk(1), vote = Yes >`
- `< w = wrk(1), vote = No >`
- `< w = wrk(2), vote = Yes >`
- `< w = wrk(2), vote = No >`

- The **scope of a variable** is the surrounding arc expressions of the transition

Transition binding enabling

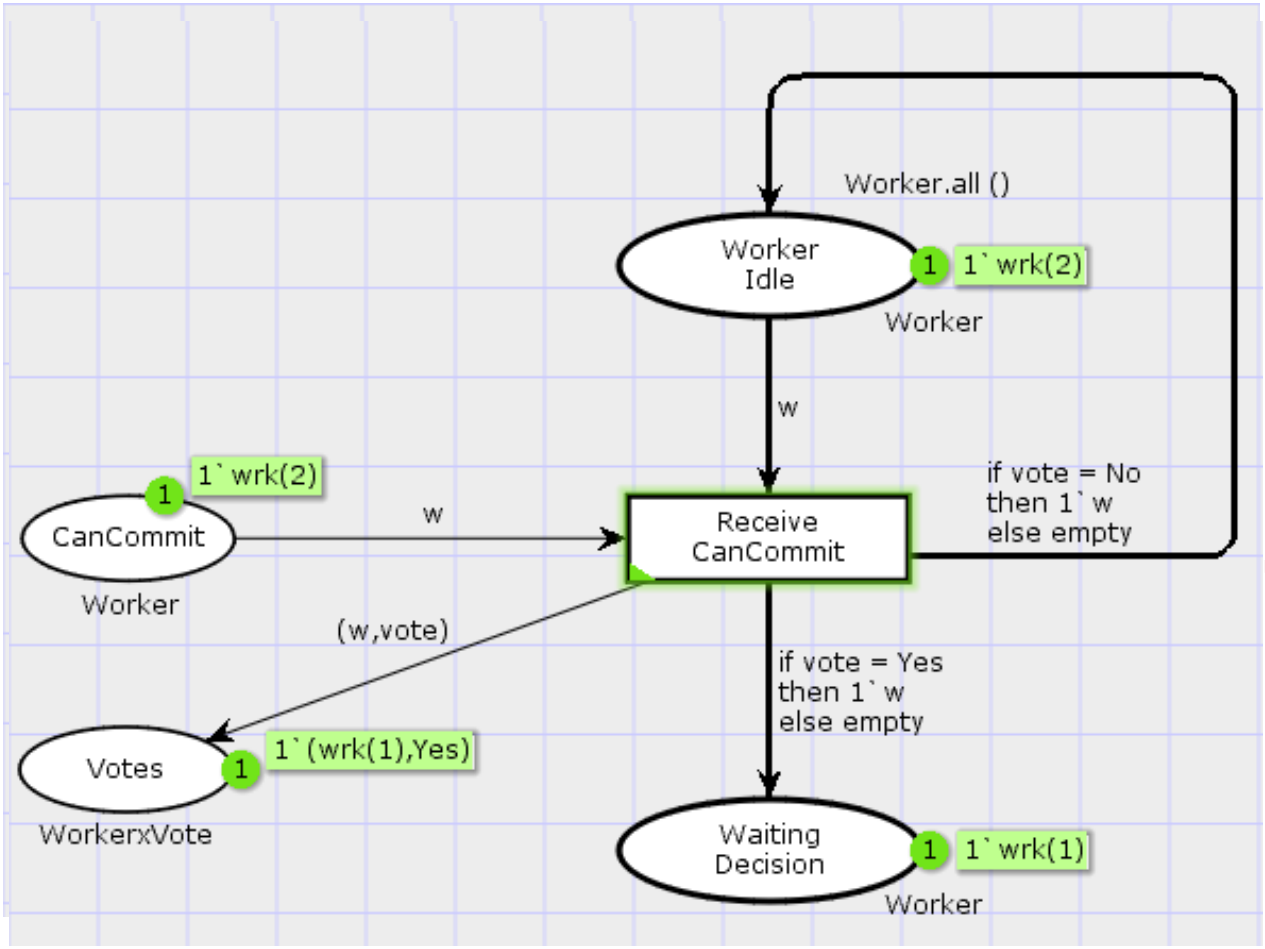
```
< w = wrk(1),vote = Yes >
```



- A transition binding is **enabled** if there are sufficient tokens on each input place
- **Tokens required on input places** are determined **by evaluating the input arc expressions in the binding** under consideration
- **Enabling condition:** the multi-set of tokens obtained must be **contained in** the multi-set of tokens present on the corresponding input place

Transition binding occurrence

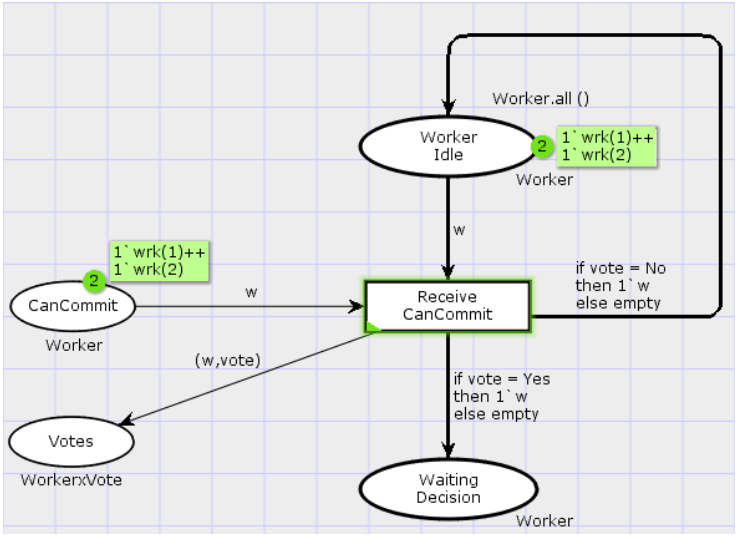
< w = wrk(1), vote = Yes >



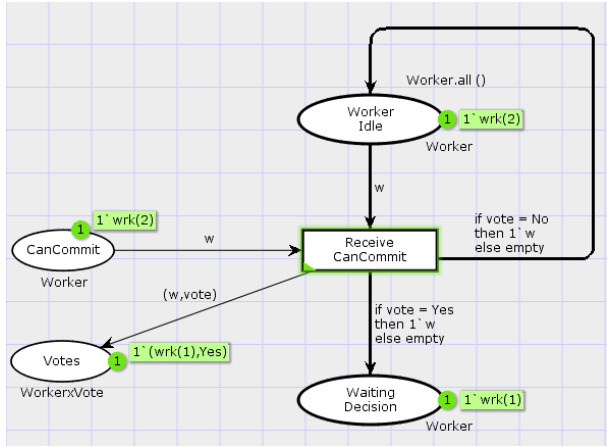
- **An enabled transition binding may occur changing the current marking (state)**
- **Tokens removed from input places: determined by evaluating the input arc expression in the binding**
- **Tokens added to output places: determined by evaluating the output arc expressions in the binding**

Binding occurrence

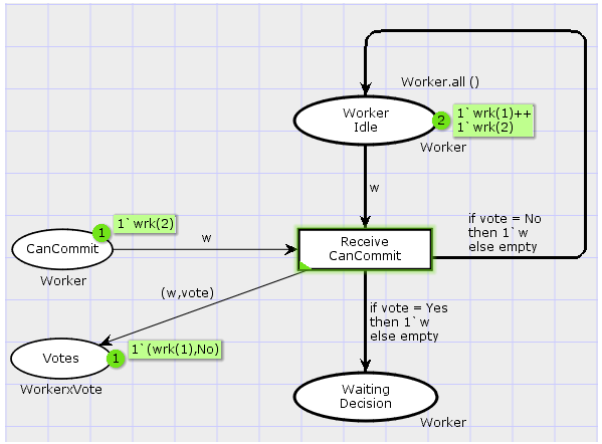
- **A transition may have several enabled bindings**



b_{1Y}



b_{1N}



Bindings

b_{1Y} = < w = wrk(1), vote = Yes >

b_{1N} = < w = wrk(1), vote = No >

b_{2Y} = < w = wrk(2), vote = Yes >

b_{2N} = < w = wrk(2), vote = No >

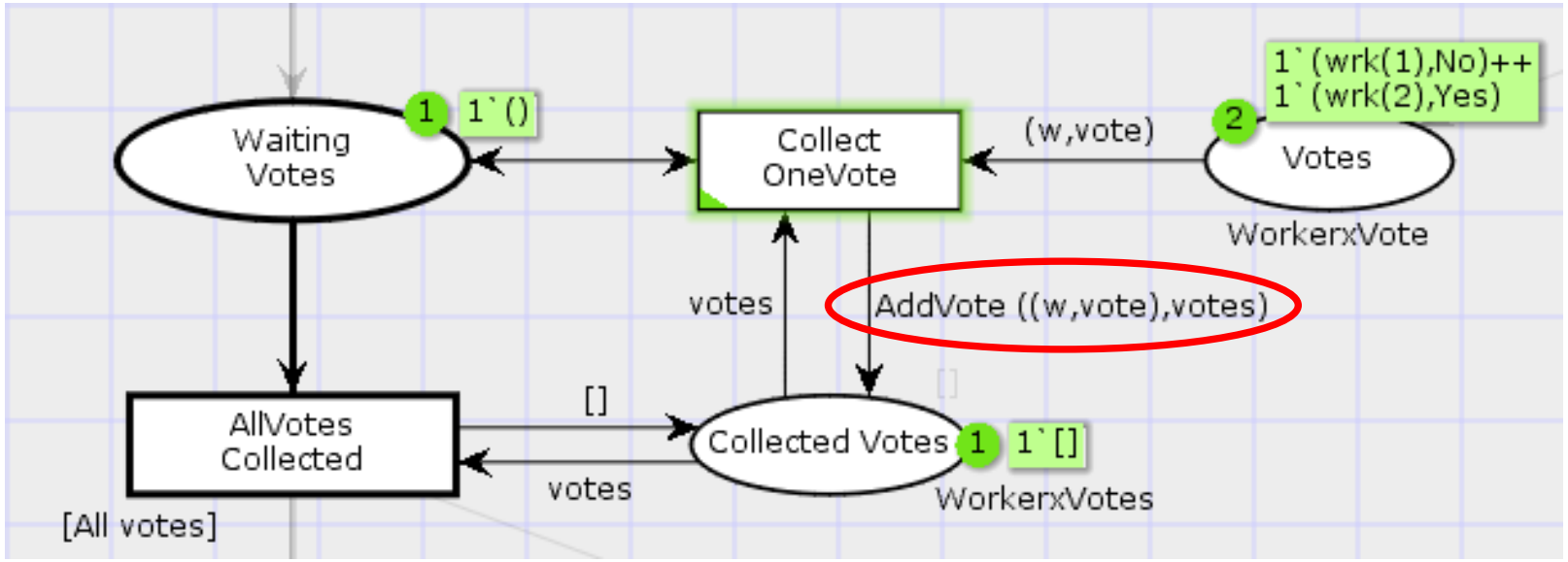
CPN Tools demo

`lecture3-cpns-simulation.cpn`

- **Simulation of CPN models**
  - Automatic simulation with visual feedback
  - Stop options and automatic simulation

Collecting votes

- **Votes are collected one at a time and accumulated in a list-token on place Votes**




```
var w     : Worker;     var vote : Vote;
colset WorkerxVote  = product Worker * Vote;
colset WorkerxVotes = list WorkerxVote;

var votes : WorkerxVotes;
```

Functions

- **The function AddVote is used to add a vote from a worker to the list of collected votes**

```
fun AddVote ((w,vote),votes) = (w,vote)::votes
```

- **Example**

```
AddVote ((wrk(1),No),[])
```

```
(wrk(1),No)::[]
```

```
[(wrk(1),No)]
```

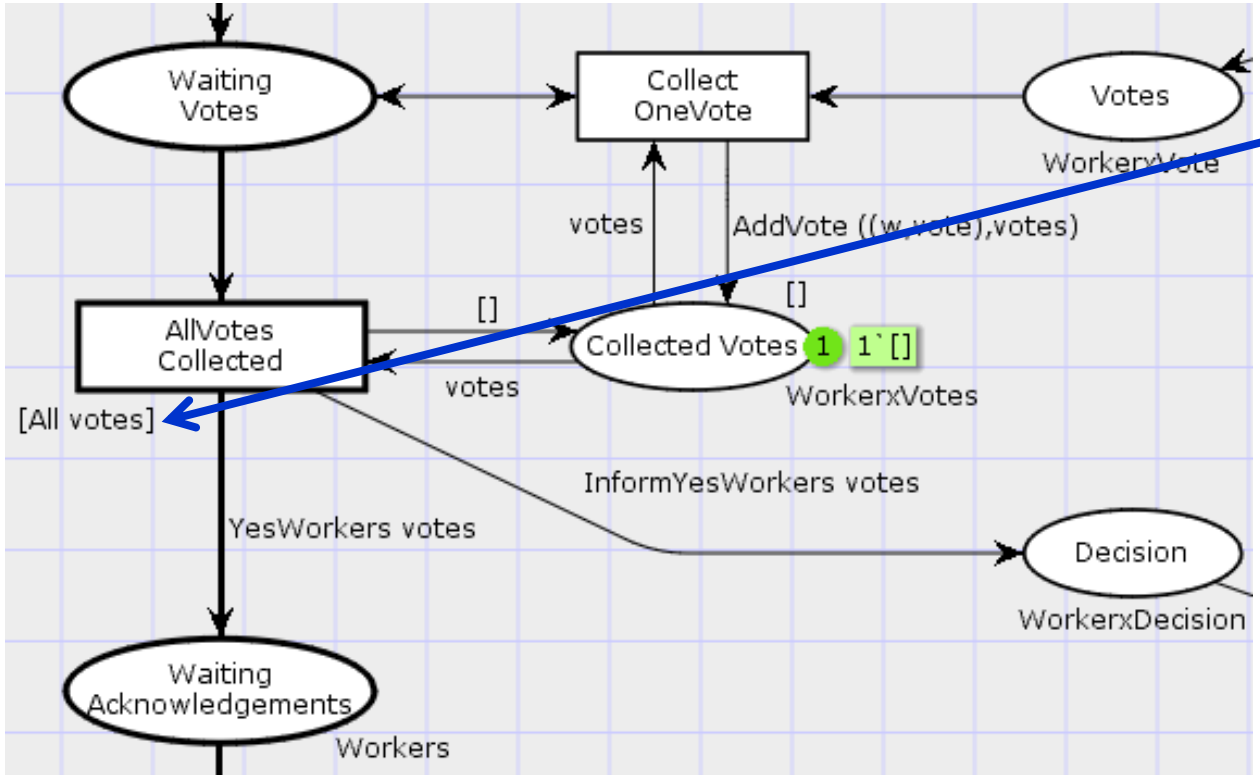
:: is the list-constructor in Standard ML

```
AddVote ((wrk(2),Yes),[(wrk(1),No)])
```

```
[(wrk(2),Yes),(wrk(1),No)]
```

Guard expressions

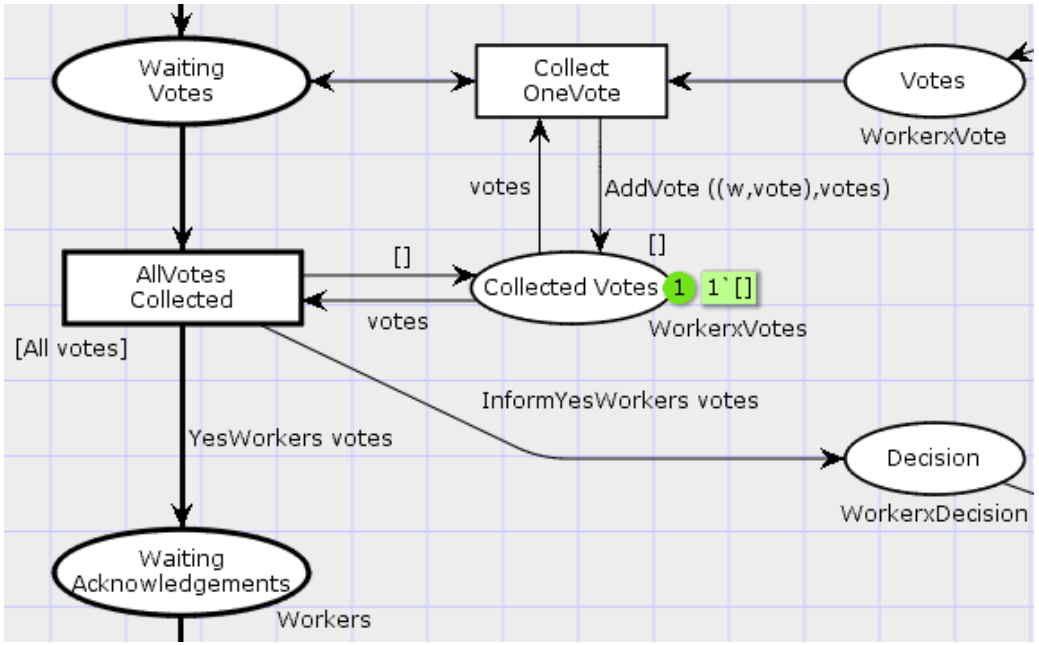
- **A transition may have a boolean guard expression which is an extra enabling condition**



The guard is by convention written in square brackets next to the transition

Collecting all votes

- **Transition AllVotesCollected should only be enabled when we have collected all votes**



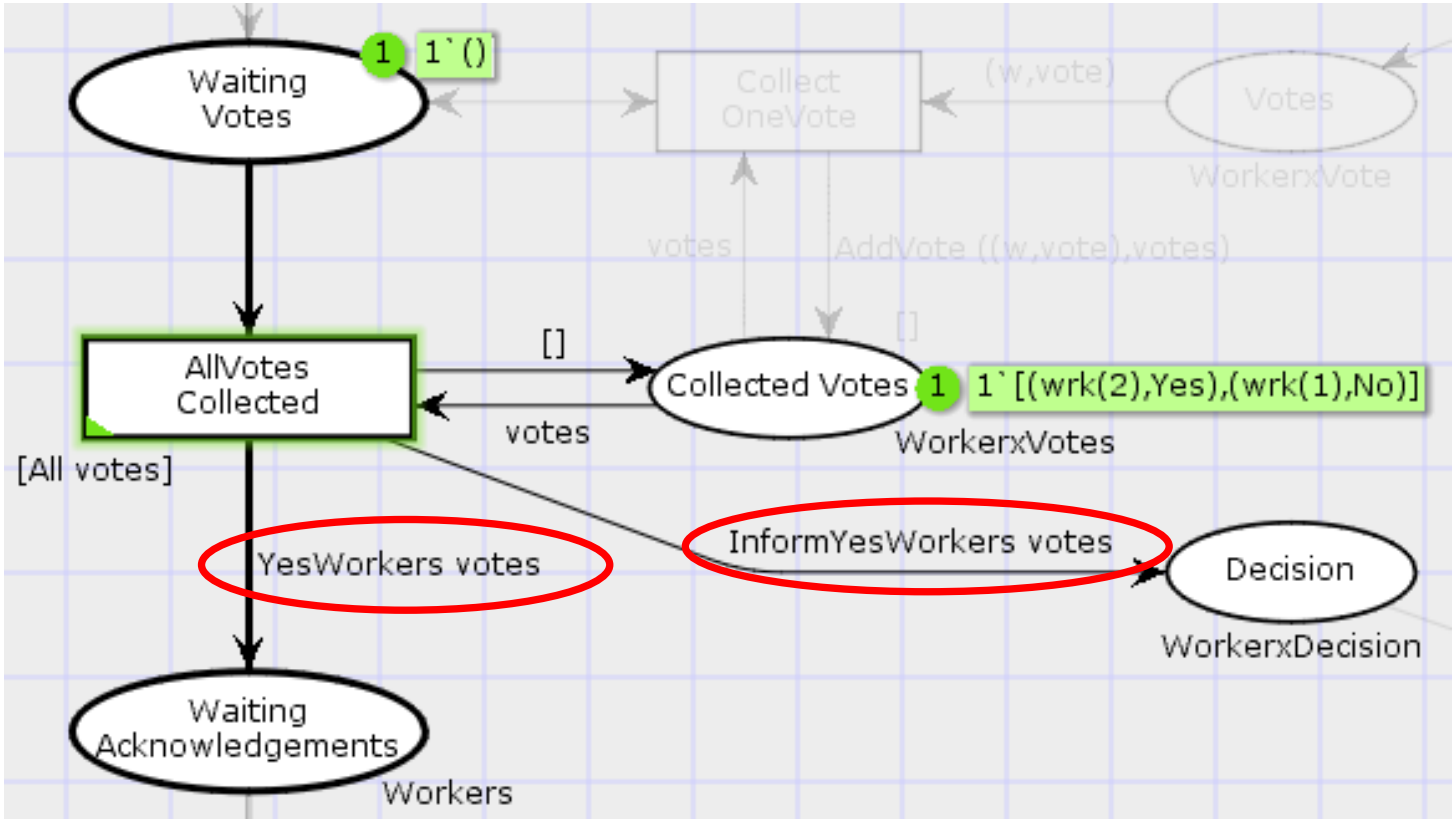

```
colset WorkerxVote =
   product Worker * Vote;

colset WorkerxVotes =
   list WorkerxVote;

var votes : WorkerxVotes
```

```
fun All votes =
(List.length votes = W)
```

`votes = [(wrk(2),Yes),(wrk(1),No)]`	`All votes`	--->	`true`
`votes = [(wrk(1),No)]`	`All votes`	--->	`false`

Informing workers

- **Functions are also used to find the workers that needs to be informed about the decision to abort or commit**



YesWorkers function

- **Obtaining the list of Yes-votes (utility function)**

```
fun yesVotes votes = List.filter
                        (fn (w,vote) => vote = Yes)
                     votes
```

```
yesVotes [(wrk(1),No),(wrk(2),Yes)]
```

```
[(wrk(2),Yes)]
```

- **Getting workers that votes Yes (projection)**

```
fun YesWorkers votes = List.map
                         (fn (w,_) => w)
                       (yesVotes votes)
```

InformYesWorkers function

```
fun InformYesWorkers votes =
  let
    val yesworkers = YesWorkers votes
    val decision =
      (if (List.length yesworkers = W)
       then commit
       else abort)
  in
    List.map (fn w => (w,decision)) yesworkers
end
```

```
InformYesWorkers [(wrk(1),No),(wrk(2),Yes)]
```

```
[(wrk(2),abort)]
```

CPN Tools demo

`lecture3-cpns-editing.cpn`

- **Starting from the simulation demo model**
  - A non-reactive version collecting the transaction result on the coordinator side
  - A new (enumeration) colour set `TransResult`
  - A new function allYes used to check if all workers voted yes
- **Editing of CPN models**
  - Incremental syntax check of the model (dependencies)
  - Adding and deleting declarations
  - Editing inscriptions
    (arc expressions, colour sets, initial markings, guards)
  - Guidelines, graphical attributes, and groups

Summary

- **Coloured Petri Nets extends Place/Transition Nets with a functional programming language**
- **Syntactical concepts**
  - Colour sets defines the data types available for modelling
  - Declaration of variables over the colour sets of the model
  - Places have a colour set determining the kind of tokens a place may contain
  - Arc expressions, initial marking- and guard expressions
- **Semantical concepts**
  - The marking of a place is a multi-set of tokens (values)
  - A binding gives values to the variables of a transition
    – scope of a variable is the surrounding arc expressions of the transition
  - Evaluation of arc expressions and guards in bindings determine enabling and the tokens removed/added by transitions