Bounded Analysis and Decomposition for Behavioural Description of Components

Jean-Claude Royer
Ecole des Mines de Nantes, OBASCO INRIA, LINA
Jean-Claude.Royer@emn.fr

Collaboration with Pascal Poizat and Gwen Salaün

FMOODS 2006, Bologna, Italy

Context

- Software component architectures

Context

- Software component architectures
- Specifications and verifications

Context

- Software component architectures
- Specifications and verifications
- Introducing complex protocols
  for expressiveness and readability

Context

- Software component architectures
- Specifications and verifications
- Introducing complex protocols
  for expressiveness and readability
- Resource and service availability properties

Context

- Software component architectures
- Specifications and verifications
- Introducing complex protocols
  for expressiveness and readability
- Resource and service availability properties
- Boundedness of dynamic systems with data

Context

- Software component architectures
- Specifications and verifications
- Introducing complex protocols
  for expressiveness and readability
- Resource and service availability properties
- Boundedness of dynamic systems with data
- Reusing classic model-checking

Plan

- Related work

Plan

- Related work
- Symbolic Transition System (STS)

Plan

- Related work
- Symbolic Transition System (STS)
- Configuration graph and interpretations

Plan

- Related work
- Symbolic Transition System (STS)
- Configuration graph and interpretations
- Boundedness of counter systems

Plan

- Related work
- Symbolic Transition System (STS)
- Configuration graph and interpretations
- Boundedness of counter systems
- Decomposition

Plan

- Related work
- Symbolic Transition System (STS)
- Configuration graph and interpretations
- Boundedness of counter systems
- Decomposition
- Examples :

Plan

- Related work
- Symbolic Transition System (STS)
- Configuration graph and interpretations
- Boundedness of counter systems
- Decomposition
- Examples :
  - The ticket protocol example

Plan

- Related work
- Symbolic Transition System (STS)
- Configuration graph and interpretations
- Boundedness of counter systems
- Decomposition
- Examples :
  - The ticket protocol example
  - The resource allocator example

Plan

- Related work
- Symbolic Transition System (STS)
- Configuration graph and interpretations
- Boundedness of counter systems
- Decomposition
- Examples :
  - The ticket protocol example
  - The resource allocator example
- Conclusion and future work

Related Work

- To complement model-checking

Related Work

- To complement model-checking
- Boundedness of generalized Petri nets (Finkel, Schnoebelen, …)

Related Work

- To complement model-checking
- Boundedness of generalized Petri nets (Finkel, Schnoebelen, ...)
- Bounded decomposition is an abstraction method (Bensalem, Clarke, Dams, ...)

Related Work

- To complement model-checking
- Boundedness of generalized Petri nets (Finkel, Schnoebelen, ...)
- Bounded decomposition is an abstraction method (Bensalem, Clarke, Dams, ...)
- Acceleration technique (Finkel and al.)

Related Work

- To complement model-checking
- Boundedness of generalized Petri nets (Finkel, Schnoebelen, ...)
- Bounded decomposition is an abstraction method (Bensalem, Clarke, Dams, ...)
- Acceleration technique (Finkel and al.)
- Theorem prover and model-checker (Rushby, ...)

Related Work

- To complement model-checking
- Boundedness of generalized Petri nets (Finkel, Schnoebelen, ...)
- Bounded decomposition is an abstraction method (Bensalem, Clarke, Dams, ...)
- Acceleration technique (Finkel and al.)
- Theorem prover and model-checker (Rushby, ...)
- Constraint programming (Delzanno and Podelsky)

Symbolic Transition System (STS)

- Component needs complex protocols with data values

Symbolic Transition System (STS)

- Component needs complex protocols with data values
- STS : a finite state and transition formalism

Symbolic Transition System (STS)

- Component needs complex protocols with data values
- STS : a finite state and transition formalism
- STS rather than automata, LTS or Petri net

Symbolic Transition System (STS)

- Component needs complex protocols with data values
- STS : a finite state and transition formalism
- STS rather than automata, LTS or Petri net
- Process algebra with values (LOTOS)
  models : STG, I/O-STS, STS ...

Symbolic Transition System (STS)

- Component needs complex protocols with data values
- STS : a finite state and transition formalism
- STS rather than automata, LTS or Petri net
- Process algebra with values (LOTOS)
  models : STG, I/O-STS, STS ...
- A transition label : `[guard] event / action`

Symbolic Transition System (STS)

- Component needs complex protocols with data values
- STS : a finite state and transition formalism
- STS rather than automata, LTS or Petri net
- Process algebra with values (LOTOS)
  models : STG, I/O-STS, STS ...
- A transition label : `[guard] event / action`
- Input (`? x`) and output (`! v`) event parameters

Symbolic Transition System (STS)

- Component needs complex protocols with data values
- STS : a finite state and transition formalism
- STS rather than automata, LTS or Petri net
- Process algebra with values (LOTOS)
  models : STG, I/O-STS, STS ...
- A transition label : `[guard] event / action`
- Input (`? x`) and output (`! v`) event parameters
- Guards : a condition to trigger the transition

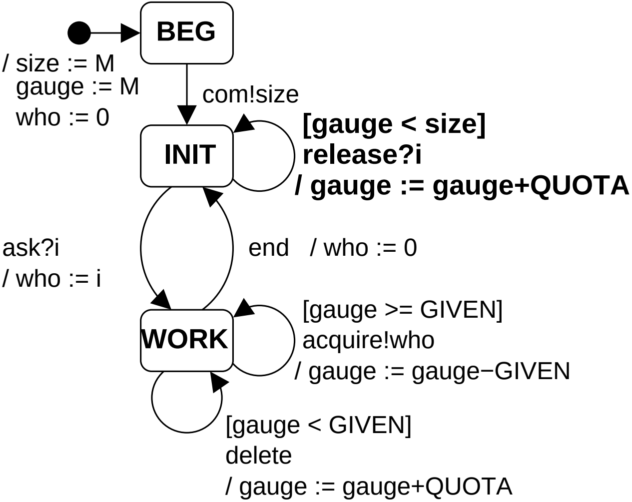
Symbolic Transition System (STS)

- Component needs complex protocols with data values
- STS : a finite state and transition formalism
- STS rather than automata, LTS or Petri net
- Process algebra with values (LOTOS)
  models : STG, I/O-STS, STS ...
- A transition label : `[guard] event / action`
- Input (`? x`) and output (`! v`) event parameters
- Guards : a condition to trigger the transition
- Action notation (imperative style in examples)

Symbolic Transition System (STS)

- Component needs complex protocols with data values
- STS : a finite state and transition formalism
- STS rather than automata, LTS or Petri net
- Process algebra with values (LOTOS)
  models : STG, I/O-STS, STS ...
- A transition label : `[guard] event / action`
- Input (`? x`) and output (`! v`) event parameters
- Guards : a condition to trigger the transition
- Action notation (imperative style in examples)
- Formal notations are provided

A Resource Allocator



Verifications Issues

- Structured synchronous product for component composition

Verifications Issues

- Structured synchronous product for component composition
- Model-checking : state explosion problems

Verifications Issues

- Structured synchronous product for component composition
- Model-checking : state explosion problems
- Several solutions for infinite state system :

Verifications Issues

- Structured synchronous product for component composition
- Model-checking : state explosion problems
- Several solutions for infinite state system :
  - Acceleration

Verifications Issues

- Structured synchronous product for component composition
- Model-checking : state explosion problems
- Several solutions for infinite state system :
  - Acceleration
  - Theorem prover

Verifications Issues

- Structured synchronous product for component composition
- Model-checking : state explosion problems
- Several solutions for infinite state system :
  - Acceleration
  - Theorem prover
  - Constraint programming

Verifications Issues

- Structured synchronous product for component composition
- Model-checking : state explosion problems
- Several solutions for infinite state system :
  - Acceleration
  - Theorem prover
  - Constraint programming
- Model-checking : efficient and automatic

Verifications Issues

- Structured synchronous product for component composition
- Model-checking : state explosion problems
- Several solutions for infinite state system :
  - Acceleration
  - Theorem prover
  - Constraint programming
- Model-checking : efficient and automatic
- Abstraction technique :

Verifications Issues

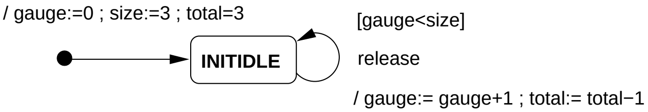
- Structured synchronous product for component composition
- Model-checking : state explosion problems
- Several solutions for infinite state system :
  - Acceleration
  - Theorem prover
  - Constraint programming
- Model-checking : efficient and automatic
- Abstraction technique :
  - Component context

Verifications Issues

- Structured synchronous product for component composition
- Model-checking : state explosion problems
- Several solutions for infinite state system :
  - Acceleration
  - Theorem prover
  - Constraint programming
- Model-checking : efficient and automatic
- Abstraction technique :
  - Component context
  - Related to boundedness

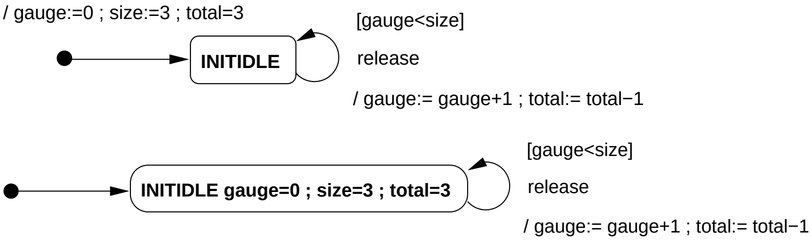
Configuration Graph

Configuration graph : unfolding receipts and data evaluation



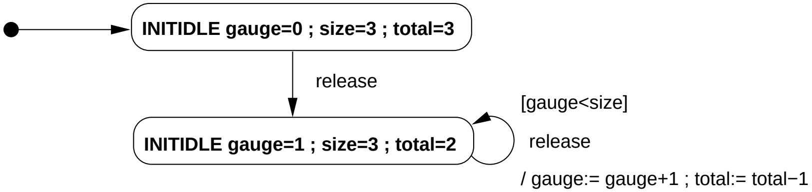
Configuration Graph

Configuration graph : unfolding receipts and data evaluation



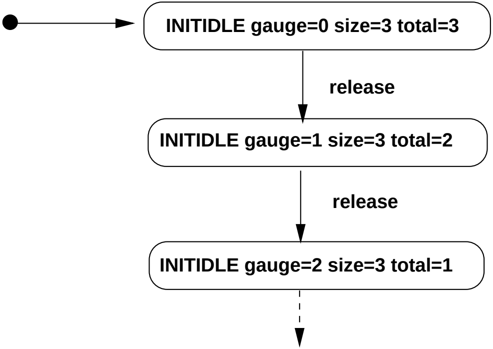
Configuration Graph

Configuration graph : unfolding receipts and data evaluation



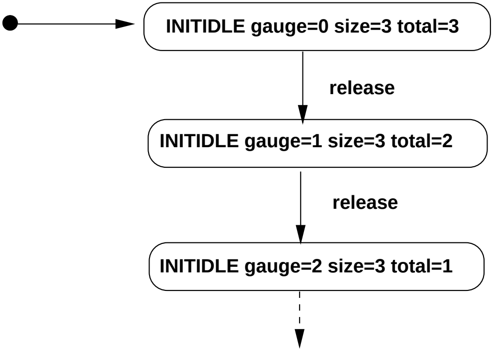
Configuration Graph

Configuration graph : unfolding receipts and data evaluation



Configuration Graph

Configuration graph : unfolding receipts and data evaluation



$G(d, v^0)$

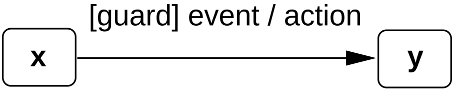
Interpretation

- The configuration graph is not necessarily a finite machine (infinite state set or infinite event set)

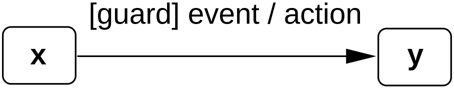
Interpretation

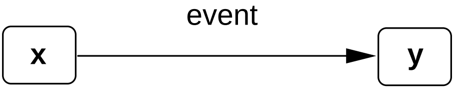
- The configuration graph is not necessarily a finite machine (infinite state set or infinite event set)
- Various LTS (I_{LTS}) interpretations of STS

Interpretation

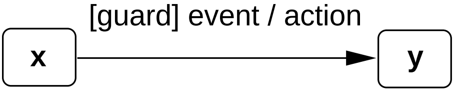
- The configuration graph is not necessarily a finite machine (infinite state set or infinite event set)
- Various LTS (I_{LTS}) interpretations of STS
- 

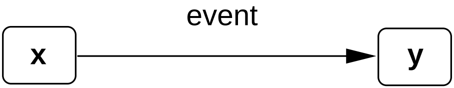
Interpretation

- The configuration graph is not necessarily a finite machine (infinite state set or infinite event set)
- Various LTS (I_{LTS}) interpretations of STS
- 

- 

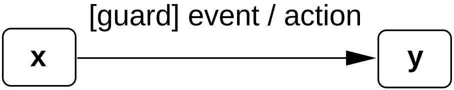
Interpretation

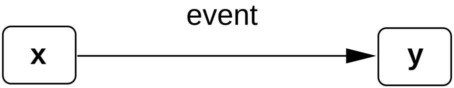
- The configuration graph is not necessarily a finite machine (infinite state set or infinite event set)
- Various LTS (I_{LTS}) interpretations of STS
- 

- 

- $I_{LTS}(d) \ \succeq \ I_{LTS}(G(d, v^0))$

Interpretation

- The configuration graph is not necessarily a finite machine (infinite state set or infinite event set)
- Various LTS (I_{LTS}) interpretations of STS
- 

- 

- $I_{LTS}(d) \ \succeq \ I_{LTS}(G(d, v^0))$
- Decomposition and boundedness

Composition

- We extend the LTS synchronous product to STS : $\otimes_V$

Composition

- We extend the LTS synchronous product to STS : $\otimes_V$
- $G(d_1 \otimes_V d_2, (v_1, v_2)) \equiv G(G(d_1, v_1) \otimes_V d_2, v_2) \equiv G(d_1, v_1) \otimes_V G(d_2, v_2)$

Composition

- We extend the LTS synchronous product to STS : $\otimes_V$
- $G(d_1 \otimes_V d_2, (v_1, v_2)) \equiv G(G(d_1, v_1) \otimes_V d_2, v_2) \equiv G(d_1, v_1) \otimes_V G(d_2, v_2)$
- More computation implies more information

Composition

- We extend the LTS synchronous product to STS : $\otimes_V$
- $G(d_1 \otimes_V d_2, (v_1, v_2)) \equiv G(G(d_1, v_1) \otimes_V d_2, v_2) \equiv G(d_1, v_1) \otimes_V G(d_2, v_2)$
- More computation implies more information
- $\ldots \succeq I_{LTS}(G(d_1, v_1) \otimes_V d_2) \succeq I_{LTS}(G(d_1 \otimes_V d_2, (v_1, v_2)))$

Composition

- We extend the LTS synchronous product to STS : $\otimes_V$
- $G(d_1 \otimes_V d_2, (v_1, v_2)) \equiv G(G(d_1, v_1) \otimes_V d_2, v_2) \equiv G(d_1, v_1) \otimes_V G(d_2, v_2)$
- More computation implies more information
- $... \succeq I_{LTS}(G(d_1, v_1) \otimes_V d_2) \succeq I_{LTS}(G(d_1 \otimes_V d_2, (v_1, v_2)))$
- $I_{LTS}(d_1 \otimes_V d_2) \succeq I_{LTS}(G(d_1, v_1) \otimes_V d_2) \succeq ...$

Composition

- We extend the LTS synchronous product to STS : $\otimes_V$
- $G(d_1 \otimes_V d_2, (v_1, v_2)) \equiv G(G(d_1, v_1) \otimes_V d_2, v_2) \equiv G(d_1, v_1) \otimes_V G(d_2, v_2)$
- More computation implies more information
- $... \succeq I_{LTS}(G(d_1, v_1) \otimes_V d_2) \succeq I_{LTS}(G(d_1 \otimes_V d_2, (v_1, v_2)))$
- $I_{LTS}(d_1 \otimes_V d_2) \succeq I_{LTS}(G(d_1, v_1) \otimes_V d_2) \succeq ...$
- Can be used to abstract some infinite and compound systems

Boundedness

- Finite resource allocation

Boundedness

- Finite resource allocation
- Finite configuration graph

Boundedness

- Finite resource allocation
- Finite configuration graph
- Semi-decidable, but decidable with Petri nets and extensions with variable transfers and constant multiplications (Finkel and al.)

Boundedness

- Finite resource allocation
- Finite configuration graph
- Semi-decidable, but decidable with Petri nets and extensions with variable transfers and constant multiplications (Finkel and al.)
- STS restrictions (counter machine) :

Boundedness

- Finite resource allocation
- Finite configuration graph
- Semi-decidable, but decidable with Petri nets and extensions with variable transfers and constant multiplications (Finkel and al.)
- STS restrictions (counter machine) :
  - Variables C_i are natural numbers

Boundedness

- Finite resource allocation
- Finite configuration graph
- Semi-decidable, but decidable with Petri nets and extensions with variable transfers and constant multiplications (Finkel and al.)
- STS restrictions (counter machine) :
  - Variables C_i are natural numbers
  - Guards are $C_i \geq M_i$

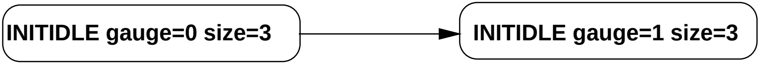
Boundedness

- Finite resource allocation
- Finite configuration graph
- Semi-decidable, but decidable with Petri nets and extensions with variable transfers and constant multiplications (Finkel and al.)
- STS restrictions (counter machine) :
  - Variables C_i are natural numbers
  - Guards are $C_i \geq M_i$
  - $C_i := \Sigma_{j=1}^{m} a_j * C_j \pm p_i$, a_j, p_i :Natural and at least one a_j is greater than 0

Boundedness

- Finite resource allocation
- Finite configuration graph
- Semi-decidable, but decidable with Petri nets and extensions with variable transfers and constant multiplications (Finkel and al.)
- STS restrictions (counter machine) :
  - Variables C_i are natural numbers
  - Guards are $C_i \geq M_i$
  - $C_i := \Sigma_{j=1}^{m} a_j * C_j \pm p_i$, a_j, p_i :Natural and at least one a_j is greater than 0
- Look for an accumulating cycle in the configuration graph

Boundedness

- Finite resource allocation
- Finite configuration graph
- Semi-decidable, but decidable with Petri nets and extensions with variable transfers and constant multiplications (Finkel and al.)
- STS restrictions (counter machine) :
  - Variables C_i are natural numbers
  - Guards are $C_i \geq M_i$
  - $C_i := \Sigma_{j=1}^{m} a_j * C_j \pm p_i$, a_j, p_i :Natural and at least one a_j is greater than 0
- Look for an accumulating cycle in the configuration graph
- 

Decomposition Principle

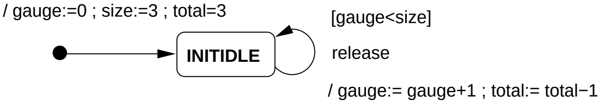
- Unfolding is split in two steps from a partition of variables

Decomposition Principle

- Unfolding is split in two steps from a partition of variables
- Data type guards and actions are decomposable in two parts

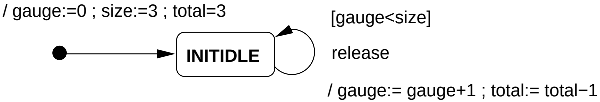
Decomposition Principle

- Unfolding is split in two steps from a partition of variables
- Data type guards and actions are decomposable in two parts

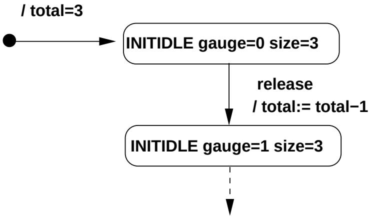


- A partition of the variables : {`gauge, size` } + {`total`}

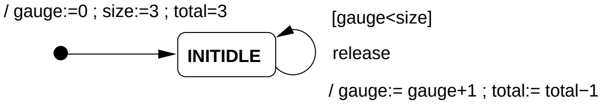
Decomposition Principle

- Unfolding is split in two steps from a partition of variables
- Data type guards and actions are decomposable in two parts
- 

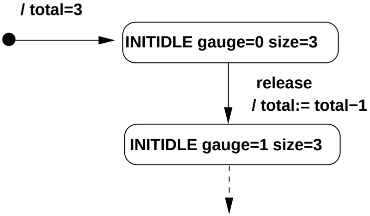
  A partition of the variables : {`gauge, size` } + {`total`}

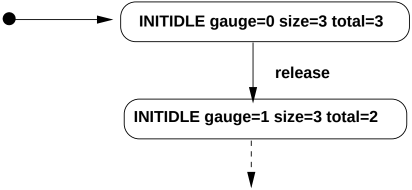


- Unfolding is split in two steps from a partition of variables
- Data type guards and actions are decomposable in two parts



- A partition of the variables : {`gauge, size` } + {`total`}





Decomposition Property

- $G_1(d, v_1^0)$ a decomposition

Decomposition Property

- $G_1(d, v_1^0)$ a decomposition
- Synchronous product is decomposable

Decomposition Property

- $G_1(d, v_1^0)$ a decomposition
- Synchronous product is decomposable
- If d, d' are decomposable STS then $d \otimes_V d'$ is decomposable

Decomposition Property

- $G_1(d, v_1^0)$ a decomposition
- Synchronous product is decomposable
- If d, d' are decomposable STS then $d \otimes_V d'$ is decomposable
- d decomposable $I_{LTS}(G_1(d, v_1^0)) \succeq I_{LTS}(G(d, v^0))$

Decomposition Property

- $G_1(d, v_1^0)$ a decomposition
- Synchronous product is decomposable
- If d, d' are decomposable STS then $d \otimes_V d'$ is decomposable
- d decomposable $I_{LTS}(G_1(d, v_1^0)) \succeq \ I_{LTS}(G(d, v^0))$
- Safety properties can be proved by simulation (Dams, Loiseaux, ...)

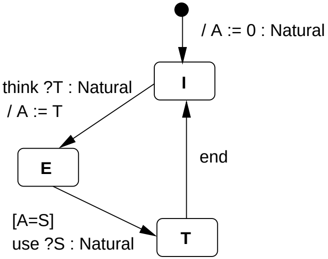
Decomposition Property

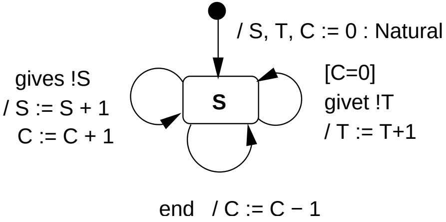
- $G_1(d, v_1^0)$ a decomposition
- Synchronous product is decomposable
- If d, d' are decomposable STS then $d \otimes_V d'$ is decomposable
- d decomposable $I_{LTS}(G_1(d, v_1^0)) \succeq \ I_{LTS}(G(d, v^0))$
- Safety properties can be proved by simulation (Dams, Loiseaux, ...)
- Bounded decomposition

Decomposition Property

- $G_1(d, v_1^0)$ a decomposition
- Synchronous product is decomposable
- If d, d' are decomposable STS then $d \otimes_V d'$ is decomposable
- d decomposable $I_{LTS}(G_1(d, v_1^0)) \succeq \ I_{LTS}(G(d, v^0))$
- Safety properties can be proved by simulation (Dams, Loiseaux, ...)
- Bounded decomposition
- They can be proved by model-checking on the bounded decomposition

Example 1 : Ticket Protocol





Process	**Server**

Synchronisations : `(think, givet), (use, gives), (end, end)`

A Bounded Decomposition

- With a finite number of processes

A Bounded Decomposition

- With a finite number of processes
- Synchronous product is not bounded

A Bounded Decomposition

- With a finite number of processes
- Synchronous product is not bounded
- Experiments with SPIN and CADP up to 6 processes

A Bounded Decomposition

- With a finite number of processes
- Synchronous product is not bounded
- Experiments with SPIN and CADP up to 6 processes
- A bounded decomposition :
  $\texttt{(\{A\}, \{C, T, S\})} = \boxed{\texttt{(\{\}, \{C\})}} + \texttt{(\{A\}, \{T, S\})}$

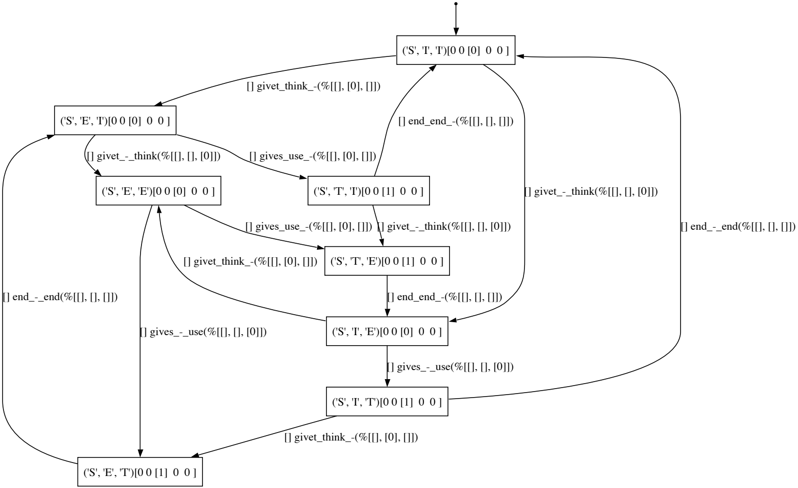
A Bounded Decomposition

- With a finite number of processes
- Synchronous product is not bounded
- Experiments with SPIN and CADP up to 6 processes
- A bounded decomposition :
  $(\{\mathtt{A}\},\ \{\mathtt{C},\ \mathtt{T},\ \mathtt{S}\}) = \boxed{(\{\},\ \{\mathtt{C}\})} + (\{\mathtt{A}\},\ \{\mathtt{T},\ \mathtt{S}\})$
- The counter choice can be assisted using communication analysis

A Bounded Decomposition

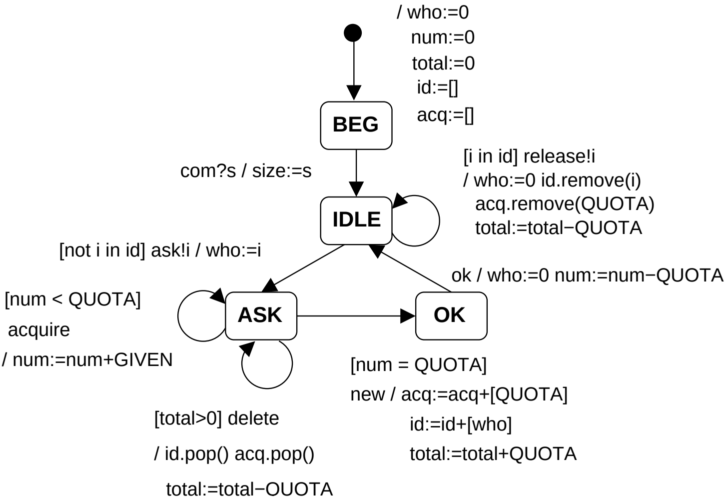
- With a finite number of processes
- Synchronous product is not bounded
- Experiments with SPIN and CADP up to 6 processes
- A bounded decomposition :
  $(\{\mathtt{A}\},\ \{\mathtt{C},\ \mathtt{T},\ \mathtt{S}\}) = \boxed{(\{\},\ \{\mathtt{C}\})} + (\{\mathtt{A}\},\ \{\mathtt{T},\ \mathtt{S}\})$
- The counter choice can be assisted using communication analysis
- One counter : boundedness is decidable !

The Bounded Analysis



Server x Process x Process

Example 2 : A Resource Allocator : the Client System



Verifications

- A partition : $\boxed{(\{\texttt{size, gauge}\}, \{\texttt{size, num, total}\})}$ $+ (\{\texttt{who}\}, \{\texttt{who, acq, id}\})$

Verifications

- A partition : $\boxed{(\{\texttt{size, gauge}\}, \{\texttt{size, num, total}\})}$ $+ (\{\texttt{who}\}, \{\texttt{who, acq, id}\})$
- Bounded decomposition (finite allocated quantities !)

Verifications

- A partition : $\boxed{(\{\texttt{size, gauge}\}, \{\texttt{size, num, total}\})}$ $+ (\{\texttt{who}\}, \{\texttt{who, acq, id}\})$
- Bounded decomposition (finite allocated quantities !)
- The sequence `delete ; delete` cannot occur

Verifications

- A partition : $\boxed{(\{\texttt{size, gauge}\}, \{\texttt{size, num, total}\})}$ $+ (\{\texttt{who}\}, \{\texttt{who, acq, id}\})$
- Bounded decomposition (finite allocated quantities !)
- The sequence `delete ; delete` cannot occur
- Decomposition deadlocks iff `GIVEN` does not divide `QUOTA`

Verifications

- A partition : $\boxed{(\{\texttt{size, gauge}\}, \{\texttt{size, num, total}\})}$ $+ (\{\texttt{who}\}, \{\texttt{who, acq, id}\})$
- Bounded decomposition (finite allocated quantities !)
- The sequence `delete ; delete` cannot occur
- Decomposition deadlocks iff `GIVEN` does not divide `QUOTA`
- Deadlock freeness of the resource allocator needs an additional property

Verifications

- A partition : $\boxed{(\{\texttt{size, gauge}\}, \{\texttt{size, num, total}\})}$ $+ (\{\texttt{who}\}, \{\texttt{who, acq, id}\})$
- Bounded decomposition (finite allocated quantities !)
- The sequence `delete ; delete` cannot occur
- Decomposition deadlocks iff `GIVEN` does not divide `QUOTA`
- Deadlock freeness of the resource allocator needs an additional property
- $\texttt{ask ; acquire}^p\texttt{ ; delete ; acquire}^r\texttt{ ; new ; ok}$ where $p + r = (\texttt{QUOTA \% GIVEN})$

Verifications

- A partition : $\boxed{(\{\texttt{size, gauge}\}, \{\texttt{size, num, total}\})}$ $+ (\{\texttt{who}\}, \{\texttt{who, acq, id}\})$
- Bounded decomposition (finite allocated quantities !)
- The sequence `delete ; delete` cannot occur
- Decomposition deadlocks iff `GIVEN` does not divide `QUOTA`
- Deadlock freeness of the resource allocator needs an additional property
- $\texttt{ask ; acquire}^p \texttt{ ; delete ; acquire}^r \texttt{ ; new ; ok}$ where $p + r = (\texttt{QUOTA \% GIVEN})$
- Bounded waiting time availability is a safety

Verifications

- A partition : $\boxed{(\{\texttt{size, gauge}\}, \{\texttt{size, num, total}\})}$ $+ (\{\texttt{who}\}, \{\texttt{who, acq, id}\})$
- Bounded decomposition (finite allocated quantities !)
- The sequence `delete ; delete` cannot occur
- Decomposition deadlocks iff `GIVEN` does not divide `QUOTA`
- Deadlock freeness of the resource allocator needs an additional property
- $\texttt{ask ; acquire}^p \texttt{ ; delete ; acquire}^r \texttt{ ; new ; ok}$ where $p + r = (\texttt{QUOTA \% GIVEN})$
- Bounded waiting time availability is a safety
- All possibly verified on the bounded decomposition

Conclusion and Future Work

- A formal component model with STS

Conclusion and Future Work

- A formal component model with STS
- Configuration graphs, boundedness, decomposition

Conclusion and Future Work

- A formal component model with STS
- Configuration graphs, boundedness, decomposition
- Other experiments : slip, bakery protocols, cash point system, ticket reservation

Conclusion and Future Work

- A formal component model with STS
- Configuration graphs, boundedness, decomposition
- Other experiments : slip, bakery protocols, cash point system, ticket reservation
- A Python prototype : STS definition, synchronous product, boundedness checking

Conclusion and Future Work

- A formal component model with STS
- Configuration graphs, boundedness, decomposition
- Other experiments : slip, bakery protocols, cash point system, ticket reservation
- A Python prototype : STS definition, synchronous product, boundedness checking
- Future work

Conclusion and Future Work

- A formal component model with STS
- Configuration graphs, boundedness, decomposition
- Other experiments : slip, bakery protocols, cash point system, ticket reservation
- A Python prototype : STS definition, synchronous product, boundedness checking
- Future work
  - Integrate other abstractions

Conclusion and Future Work

- A formal component model with STS
- Configuration graphs, boundedness, decomposition
- Other experiments : slip, bakery protocols, cash point system, ticket reservation
- A Python prototype : STS definition, synchronous product, boundedness checking
- Future work
  - Integrate other abstractions
  - Automate using communication analysis, data isomorphism, ...

Conclusion and Future Work

- A formal component model with STS
- Configuration graphs, boundedness, decomposition
- Other experiments : slip, bakery protocols, cash point system, ticket reservation
- A Python prototype : STS definition, synchronous product, boundedness checking
- Future work
  - Integrate other abstractions
  - Automate using communication analysis, data isomorphism, ...
  - Optimise the prototype