Historique

- A l'origine les objets ...
- Essais de rationalisation (OMT, UML)
- Les types abstraits graphiques
- Technique de spécifications formelles mixtes (Korrigan)
- Moyens de preuve associés
- Notamment preuve de propriétés dynamiques

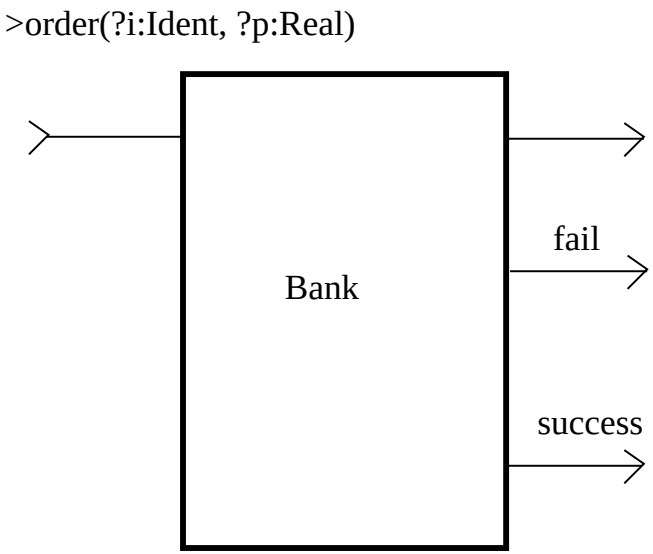
Objectifs (perso)

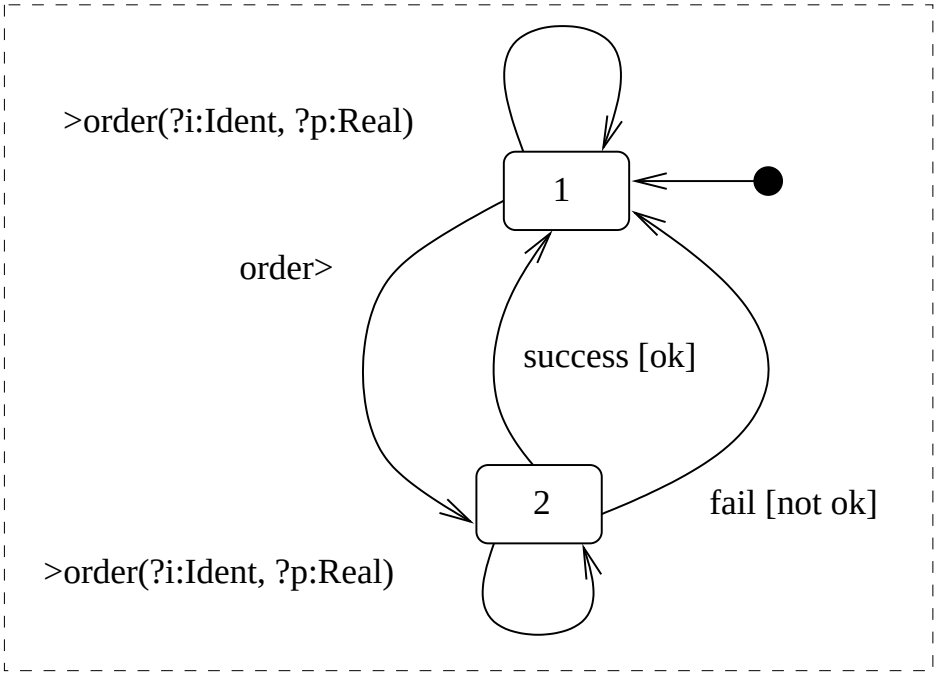
- S’intéresser aux aspects formels des composants (ADL)
- Outils d’analyse, de preuve, de contrôle etc
- Montrer un lien assez direct avec un codage objet
- Communication synchrone et asynchrone
- Expliciter le comportement dynamique des composants et des architectures

Model Principles (1/2)

- Composant (type de) atomique ou composite
- Notion d'interface (des ports de communication) classique
- Comportement dynamique explicite avec des Systèmes de Transition Symboliques (STS)
- Des descriptions graphiques, de la "lisibilité"
- Un système de réservation de billet avec 4 composants

Le composant Bank





Type Asbtrait Graphique

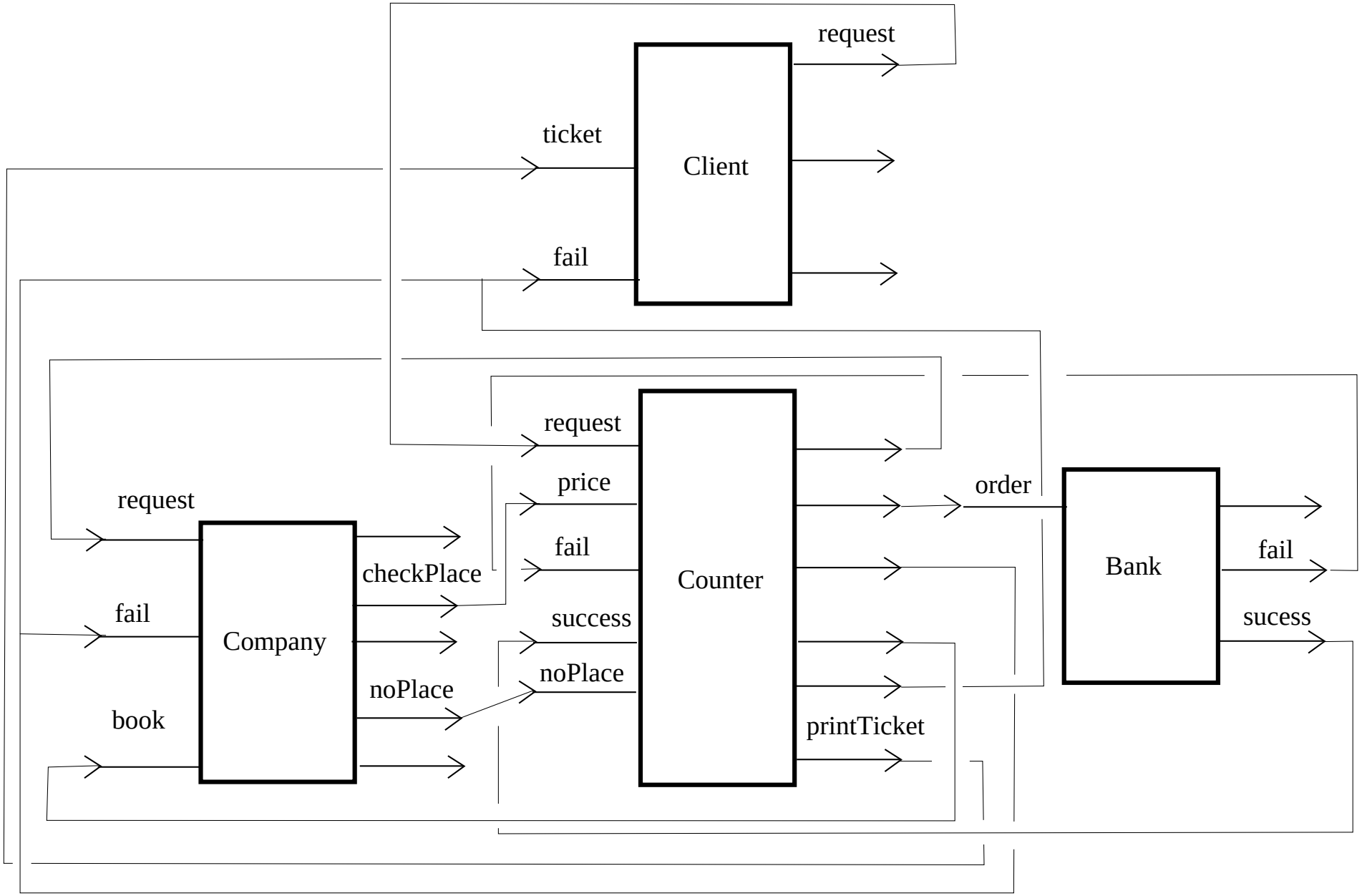
- Interprétation des transitions comme des fonctions d'un TAD

$$\{P1(self)\}\ \texttt{order>(self,i,p)}\ \{P2(self)\}$$

- STS = interprétation abstraite (relation d'équivalence partielle) sur le TAD
- Prédicats d'états, pré-conditions, prédicat de définition
- Génération automatique à partir du STS
- Exclusivité et complémentarité des prédicats d'états

Une architecture graphique

- Séparation des communications des composants
- Noms locaux des services
- Une colle pour définir les connections de l'architecture
- En fait une forme de logique temporelle
- Un modèle sémantique basé sur la communication lors de la synchronisation
- Dynamicité des communications grâce aux gardes


request
ticket
Client
fail
request
price
fail
success
noPlace
Counter
printTicket
order
Bank
fail
sucess
request
fail
book
Company
checkPlace
noPlace

Comportement dynamique global

- Comment avoir une idée du comportement global du système ?
- Produit synchronisé des STSs
- L'architecture donne les règles de synchronisation
- Le résultat est un STS avec des transitions et des états structurés
- Propriétés algébriques pour le produit

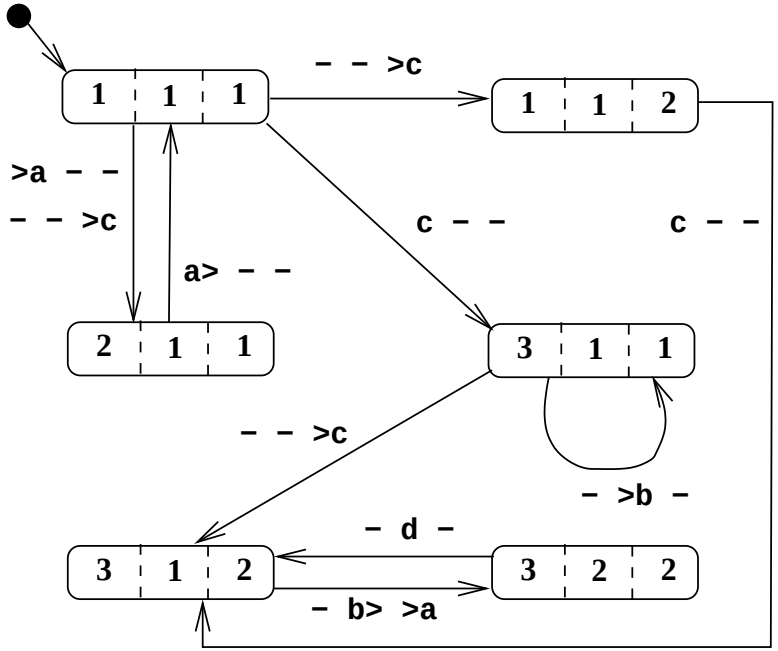
Technique de preuve

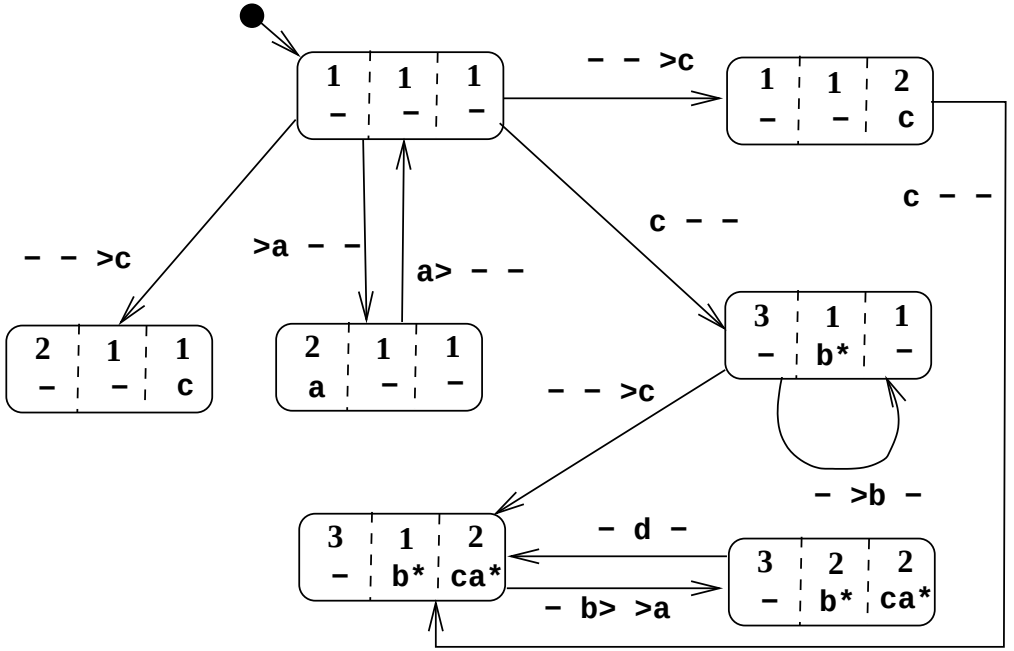
- Expérimentations avec Larch Prover et maintenant PVS
- Cadre de preuve générale mais collaboration possible avec des techniques automatiques plus spécialisées
- Une valeur du TAD = un chemin de constructeur = un historique du système
- Donc des preuves de propriétés temporelles sont possibles
- Extension stricte de CTL* avec des données

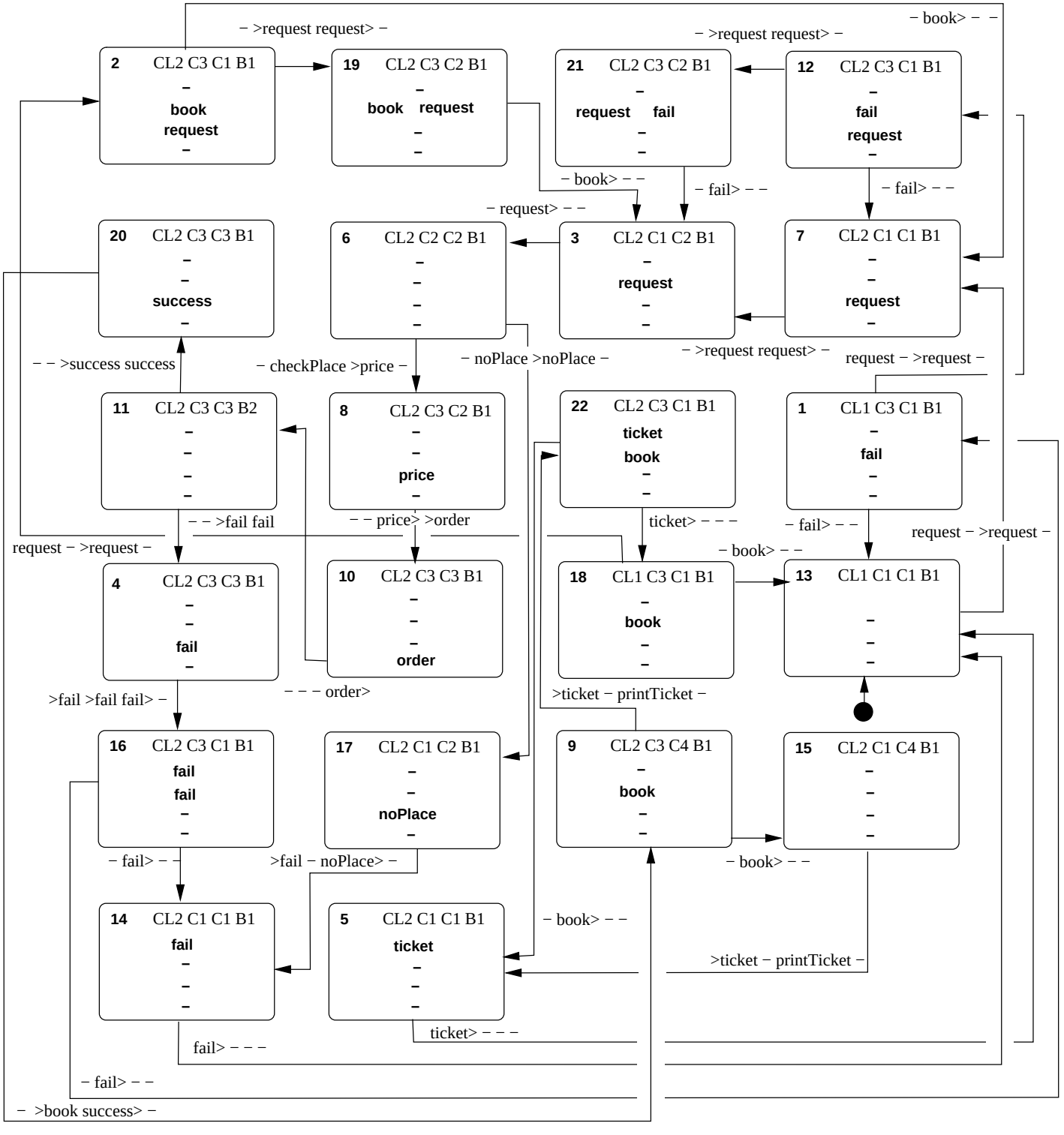
Calcul de la taille des boîtes à lettres

- Communications asynchrones
- On s'intéresse spécifiquement à certaines gardes : [&op] and [not fullMailBox]
  - `width` : une simulation bornée
  - `bound` : un calcul exhaustif des configurations
  - `...` : un algorithme additionnel peut prendre en compte les capacités finies
- +++

Un exemple fictif






2 CL2 C3 C1 B1
book
request
19 CL2 C3 C2 B1
book request
21 CL2 C3 C2 B1
request fail
12 CL2 C3 C1 B1
fail
request
20 CL2 C3 C3 B1
success
6 CL2 C2 C2 B1
3 CL2 C1 C2 B1
request
7 CL2 C1 C1 B1
request
11 CL2 C3 C3 B2
8 CL2 C3 C2 B1
price
22 CL2 C3 C1 B1
ticket
book
1 CL1 C3 C1 B1
fail
4 CL2 C3 C3 B1
fail
10 CL2 C3 C3 B1
order
18 CL1 C3 C1 B1
book
13 CL1 C1 C1 B1
16 CL2 C3 C1 B1
fail
fail
17 CL2 C1 C2 B1
noPlace
9 CL2 C3 C4 B1
book
15 CL2 C1 C4 B1
14 CL2 C1 C1 B1
fail
5 CL2 C1 C1 B1
ticket
− book> − −
− >request request> −
− request> − −
− fail> − −
− >success success
− checkPlace >price −
− noPlace >noPlace −
request − >request −
− − >fail fail
− − price> >order
ticket> − − −
− − − order>
>ticket − printTicket −
>fail >fail fail> −
>fail − noPlace> −
− book> − −
ticket> − − −
fail> − − −
− fail> − −
− >book success> −

To Analyse the Global Behaviour

- Model-checking

To Analyse the Global Behaviour

- Model-checking
- Proving

To Analyse the Global Behaviour

- Model-checking

Automatic but needs a finite state machine

- Proving

To Analyse the Global Behaviour

- Model-checking

Automatic but needs a finite state machine
Bound data types, but we loose abstraction and readability

- Proving

To Analyse the Global Behaviour

- Model-checking

Automatic but needs a finite state machine
Bound data types, but we loose abstraction and readability

- Proving

Generally not automatic but more powerful

To Analyse the Global Behaviour

- Model-checking

Automatic but needs a finite state machine
Bound data types, but we loose abstraction and readability

- Proving

Generally not automatic but more powerful
Conjunction with model-checking

To Analyse the Global Behaviour

- Model-checking

Automatic but needs a finite state machine
Bound data types, but we loose abstraction and readability

- Proving

Generally not automatic but more powerful
Conjunction with model-checking

$\Longrightarrow$ A general approach (prover) and automatic specialized tools (algorithms, model-checking, ...)