UML&FM 2012

27th August 2012
Paris, France

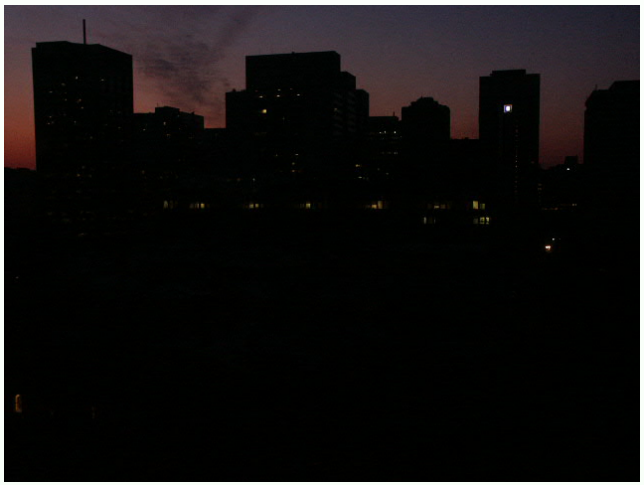
Formalizing Non-Concurrent UML State Machines Using Colored Petri Nets

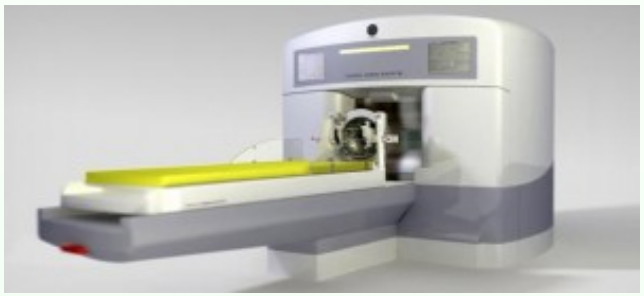
Étienne André, Christine Choppy, Kaïs Klaï

Laboratoire d'Informatique de Paris Nord
Université Paris 13, Sorbonne Paris Cité, France

Context: Verification of Complex Systems

- Need for early bug detection
  - Bugs discovered when final testing: expensive
  - Need for a thorough modeling phase

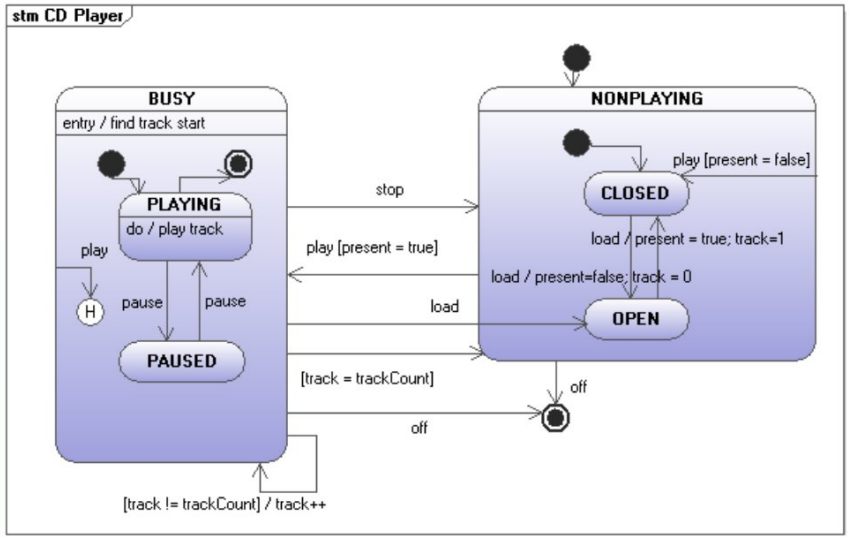




UML Behavioral State Machines

- Transition systems used to express the behavior of dynamic systems
- Specified in [OMG, 2009] (version 2.2)
- Widely used in the industry
- Semantics not formally expressed
  - Informal specification in [OMG, 2009]
  - Not directly suitable for formal methods

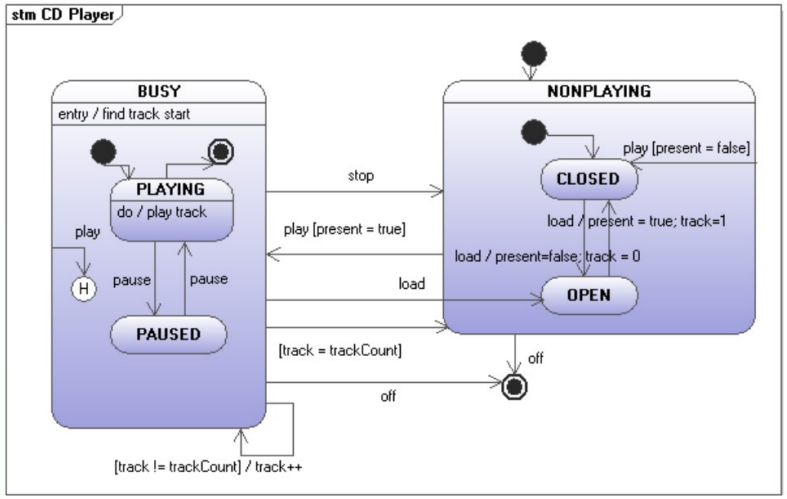
Example of a CD Player [Zhang and Liu, 2010]



- Features
  - A hierarchy of simple and composite states
  - Transitions (including inter-level) with events
  - Entry (`find track start`) and do (`play track`) behaviors
  - Global variables (`present` and `track`)
  - History pseudostate (`H`)

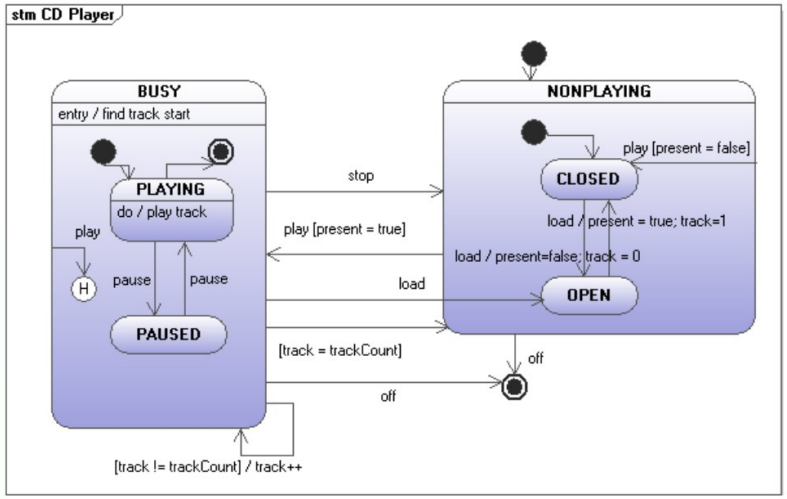
Example of a CD Player (cont.)

- This example is simple
  - Few states, few events, few variables
  - No concurrency
  - No exit behavior



Example of a CD Player (cont.)

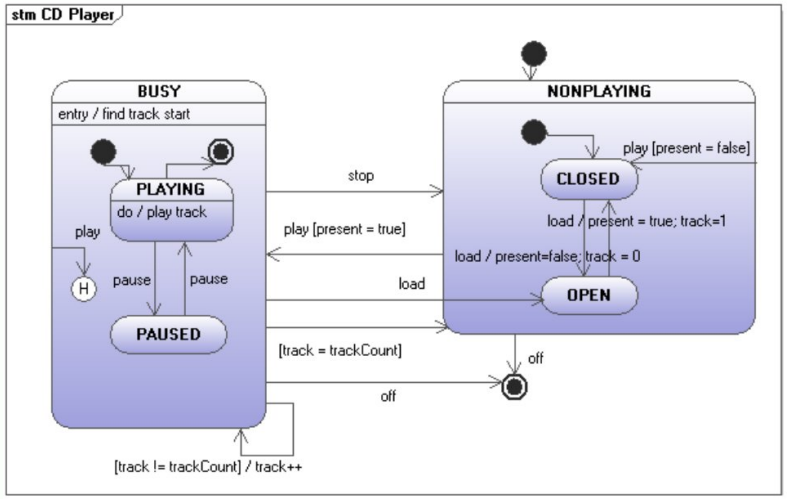
- This example is simple
  - Few states, few events, few variables
  - No concurrency
  - No exit behavior



- And still... Can we ensure the following?
  - "When in **PLAYING**, there is a CD in the player"
  - "When in **PLAYING**, the track number is always between 1 and `trackCount`"

Example of a CD Player (cont.)

- This example is simple
  - Few states, few events, few variables
  - No concurrency
  - No exit behavior



- And still... Can we ensure the following?
  - "When in **PLAYING**, there is a CD in the player"
  - "When in **PLAYING**, the track number is always between 1 and `trackCount`"

- Not easy to guarantee!
  (So what about larger case studies...)

Main Goal

Goal

"Give a semantics to UML state machines so as to allow for formal verification."

Main Goal

Goal

"Give a semantics to UML state machines so as to allow for formal verification."

- We choose here to automatically translate UML state machines to colored Petri nets (CPNs)
- Set of considered constructs
  - Hierarchy of composite states
  - Inter-level transitions
  - Entry, do, exit behaviors with global variables
  - History pseudostates
  - No concurrency (no fork, join, synchronization)

Related Works

- Semantics directly defined
  - [Jin et al., 2004, Dubrovin and Junttila, 2007]
- Translations of ULM state machines to CPNs
  - [Pettit IV and Gomaa, 2006, Lian et al., 2008, Choppy et al., 2011]
- Translations of ULM state machines to other formalisms
  - SPIN [Latella et al., 1999], SMV [Clarke and Heinle, 2000], CSP# [Zhang and Liu, 2010], etc.

Related Works

- Semantics directly defined
  - [Jin et al., 2004, Dubrovin and Junttila, 2007]
- Translations of ULM state machines to CPNs
  - [Pettit IV and Gomaa, 2006, Lian et al., 2008, Choppy et al., 2011]
- Translations of ULM state machines to other formalisms
  - SPIN [Latella et al., 1999], SMV [Clarke and Heinle, 2000], CSP# [Zhang and Liu, 2010], etc.

- Limitations
  - Almost always restrictive subset of syntactic constructs
  - Often limited specification of properties

Outline

1. Colored Petri Nets
2. Translation
3. Application to the CD Player
4. Perspectives

Outline

1 Colored Petri Nets

2 Translation

3 Application to the CD Player

4 Perspectives

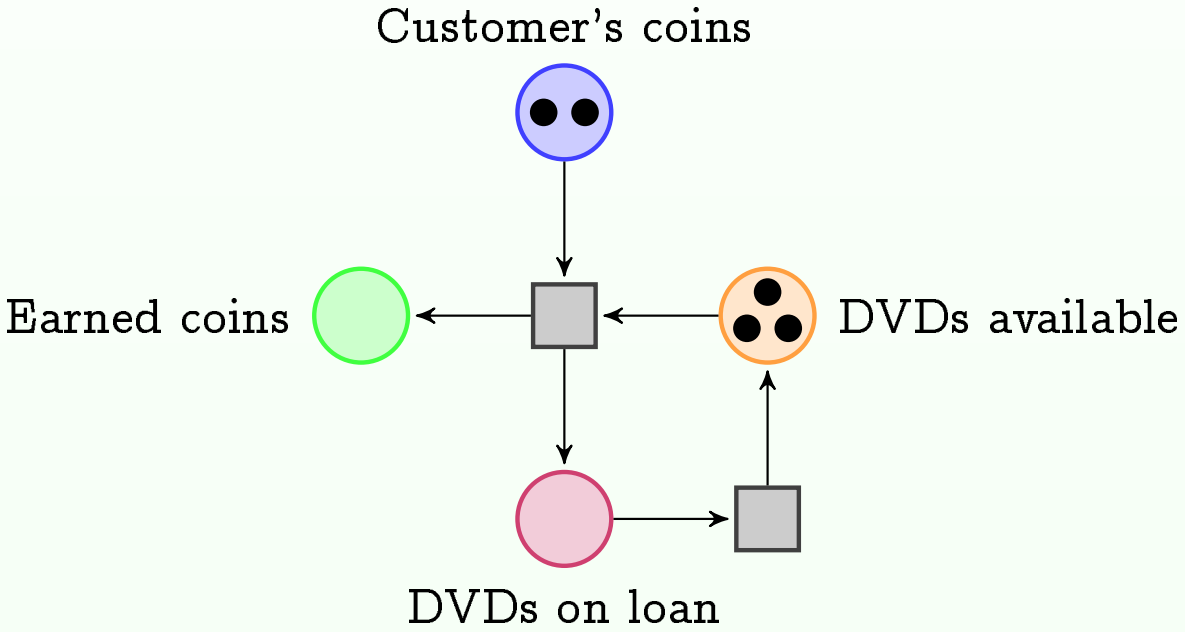
Petri Nets [Petri, 1962]

- Advantages of Petri nets
  - Detailed view of the process with an expressive graphical representation
  - A formal semantics
  - Powerful tools to test and check the model

Petri Nets [Petri, 1962]

- Advantages of Petri nets
  - Detailed view of the process with an expressive graphical representation
  - A formal semantics
  - Powerful tools to test and check the model

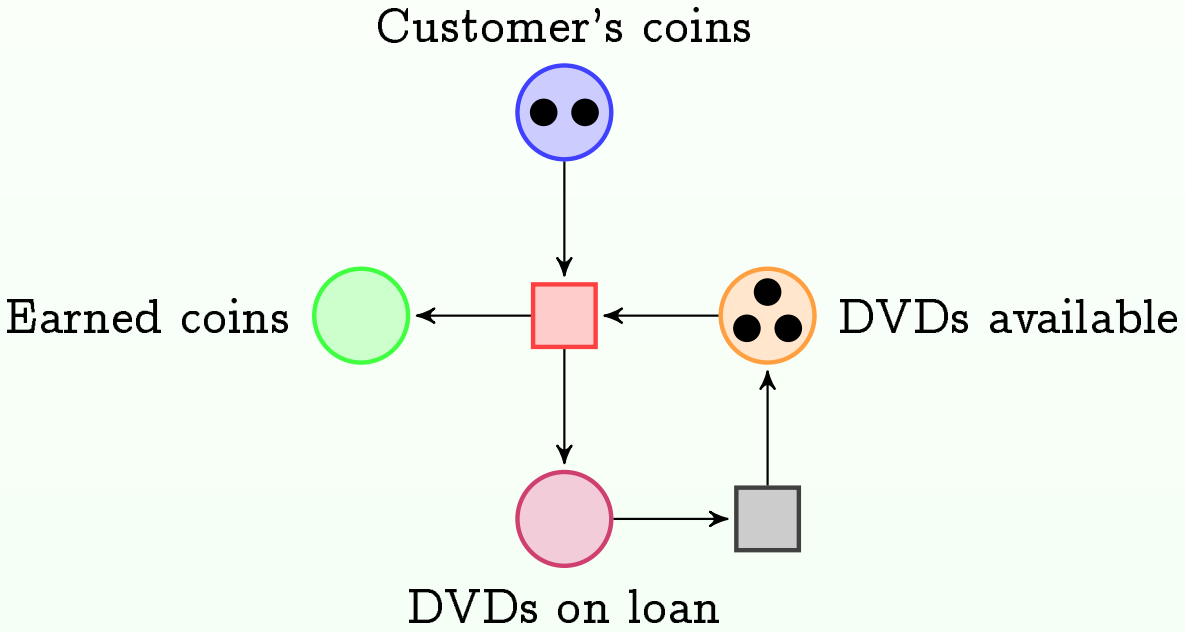
- Example: A DVD renting machine



Petri Nets [Petri, 1962]

- Advantages of Petri nets
  - Detailed view of the process with an expressive graphical representation
  - A formal semantics
  - Powerful tools to test and check the model

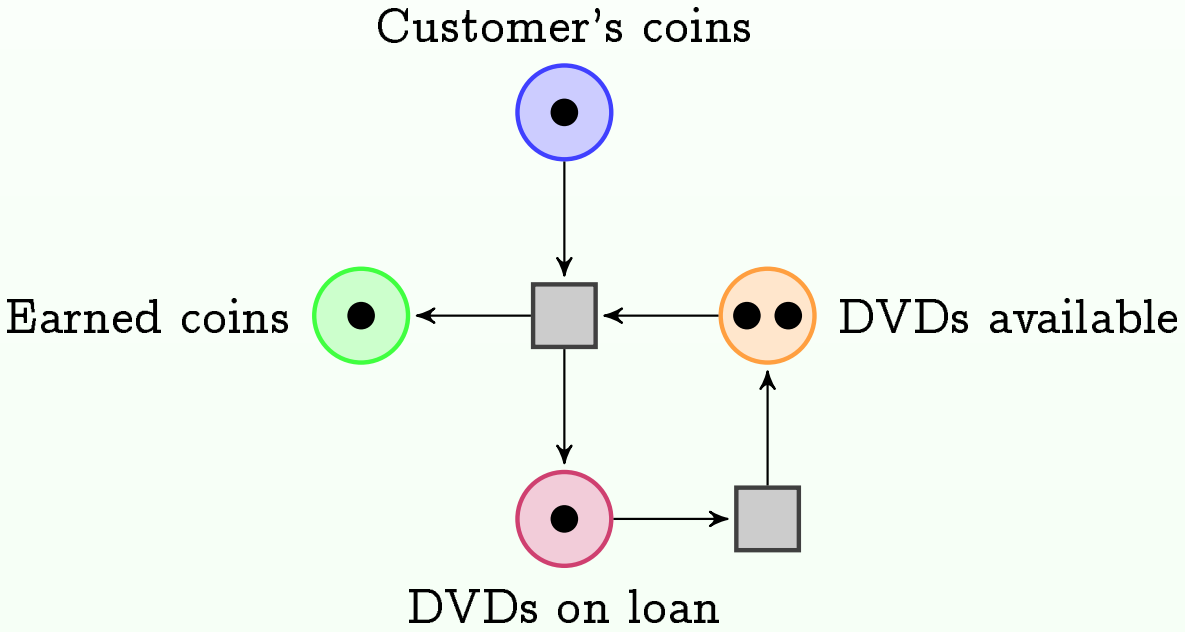
- Example: A DVD renting machine



Petri Nets [Petri, 1962]

- Advantages of Petri nets
  - Detailed view of the process with an expressive graphical representation
  - A formal semantics
  - Powerful tools to test and check the model

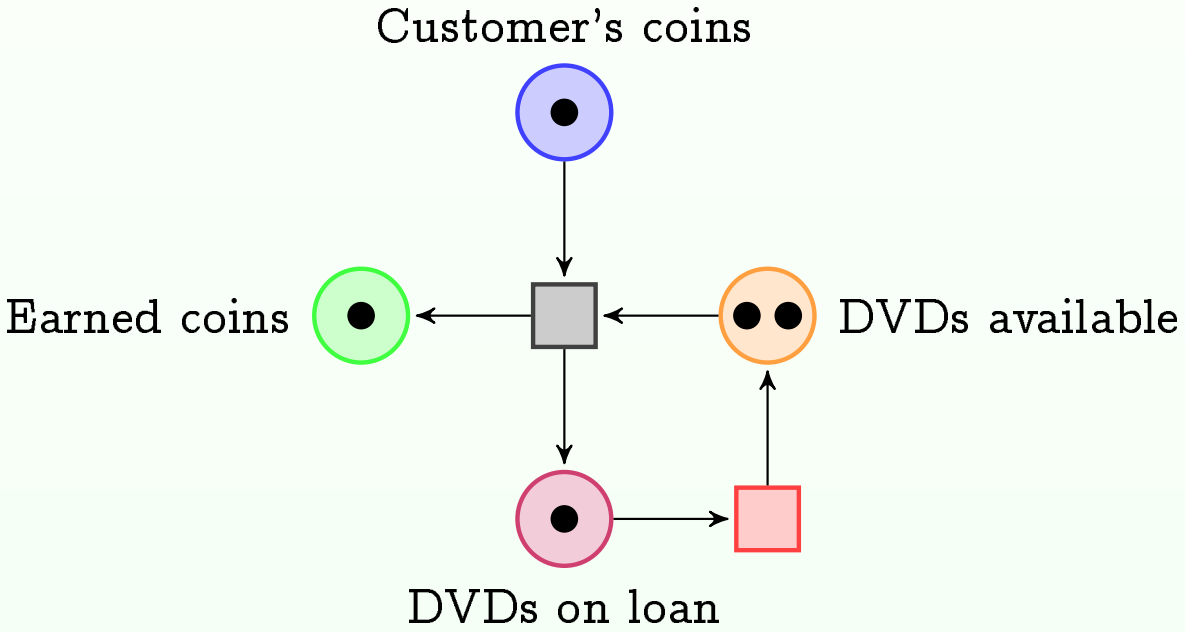
- Example: A DVD renting machine



Petri Nets [Petri, 1962]

- Advantages of Petri nets
  - Detailed view of the process with an expressive graphical representation
  - A formal semantics
  - Powerful tools to test and check the model

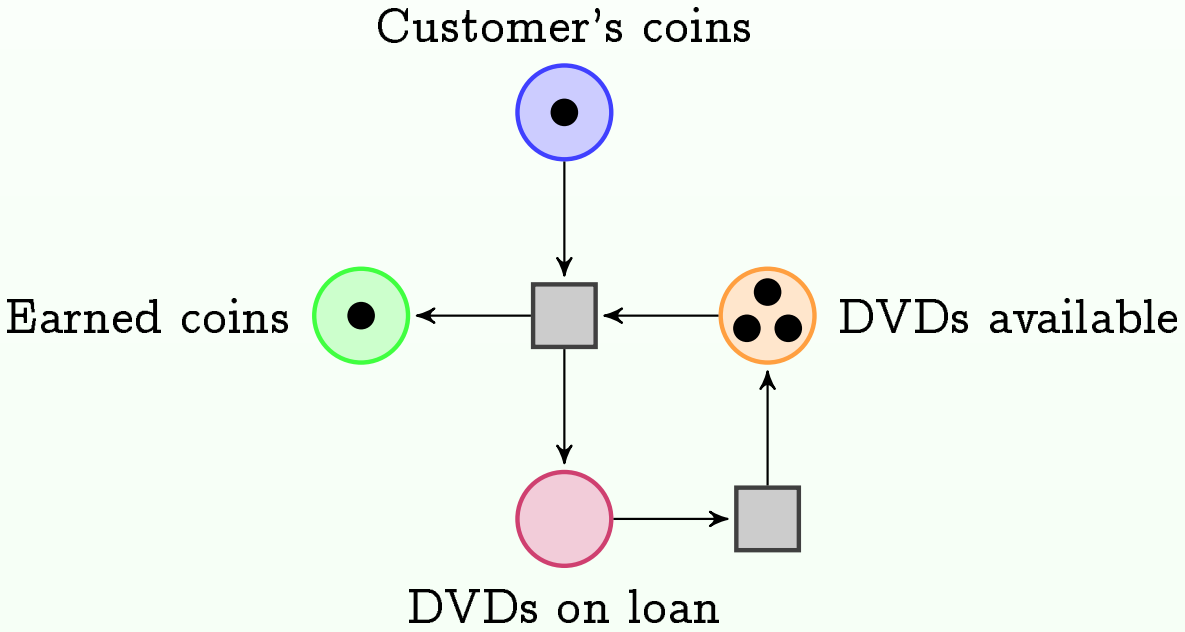
- Example: A DVD renting machine



Petri Nets [Petri, 1962]

- Advantages of Petri nets
  - Detailed view of the process with an expressive graphical representation
  - A formal semantics
  - Powerful tools to test and check the model

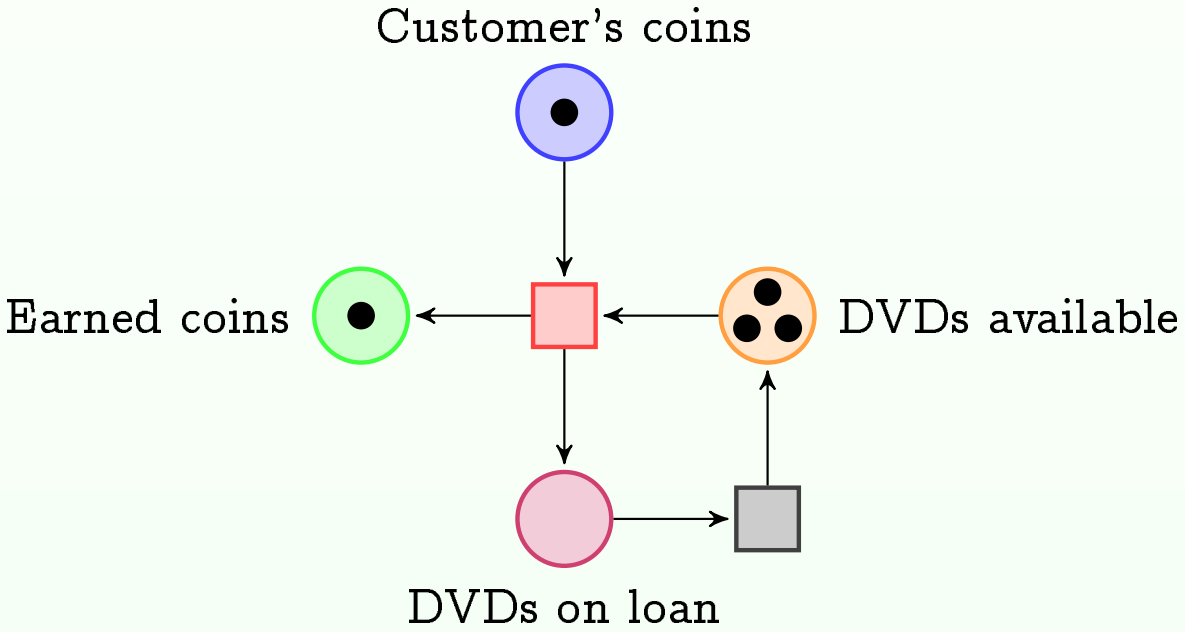
- Example: A DVD renting machine



Petri Nets [Petri, 1962]

- Advantages of Petri nets
  - Detailed view of the process with an expressive graphical representation
  - A formal semantics
  - Powerful tools to test and check the model

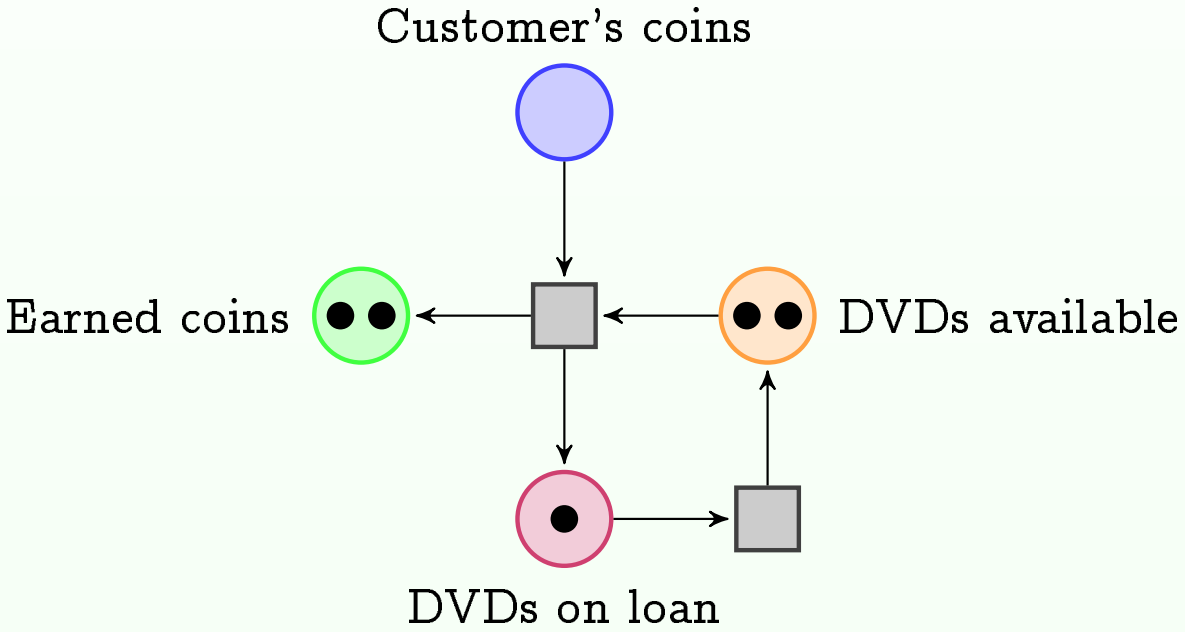
- Example: A DVD renting machine



Petri Nets [Petri, 1962]

- Advantages of Petri nets
  - Detailed view of the process with an expressive graphical representation
  - A formal semantics
  - Powerful tools to test and check the model

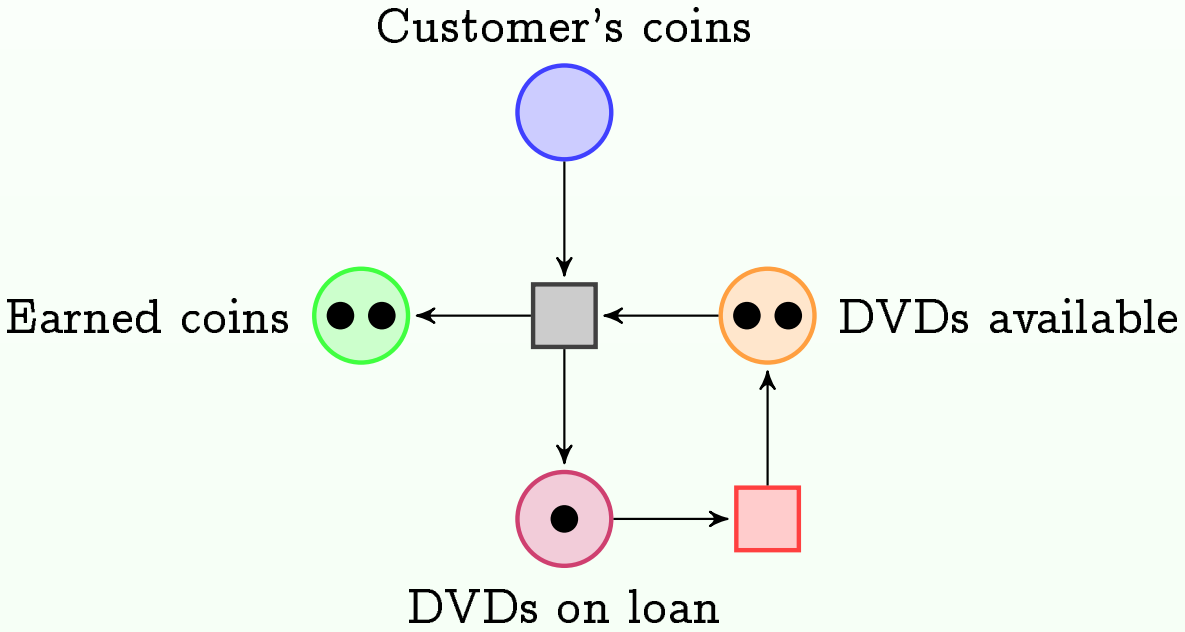
- Example: A DVD renting machine



Petri Nets [Petri, 1962]

- Advantages of Petri nets
  - Detailed view of the process with an expressive graphical representation
  - A formal semantics
  - Powerful tools to test and check the model

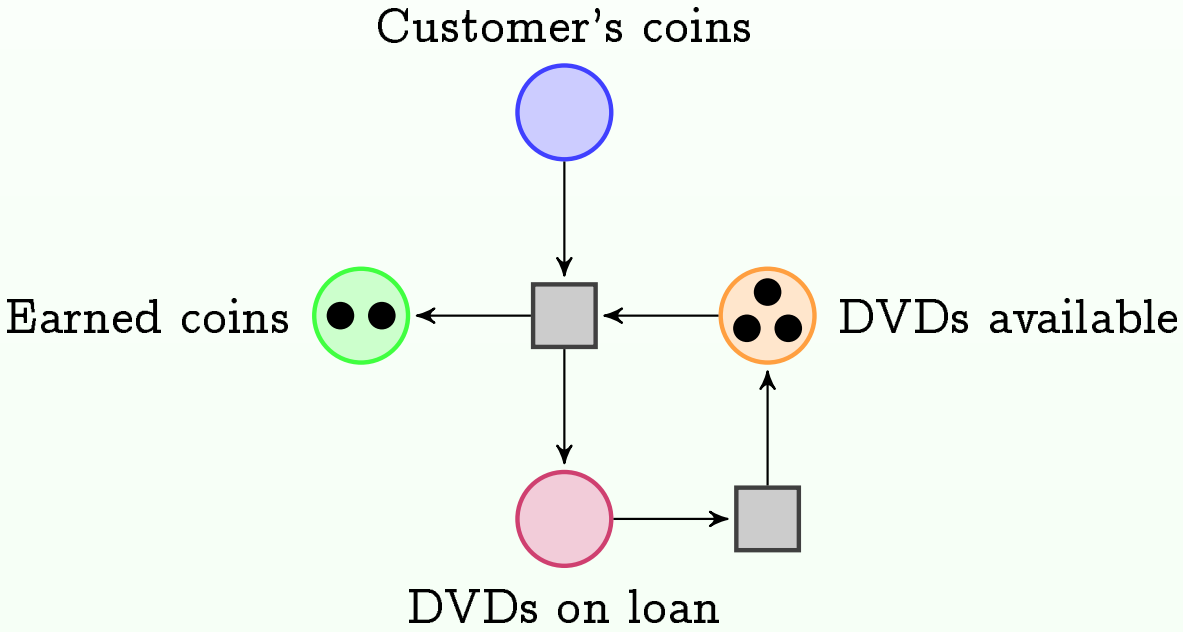
- Example: A DVD renting machine



Petri Nets [Petri, 1962]

- Advantages of Petri nets
  - Detailed view of the process with an expressive graphical representation
  - A formal semantics
  - Powerful tools to test and check the model

- Example: A DVD renting machine



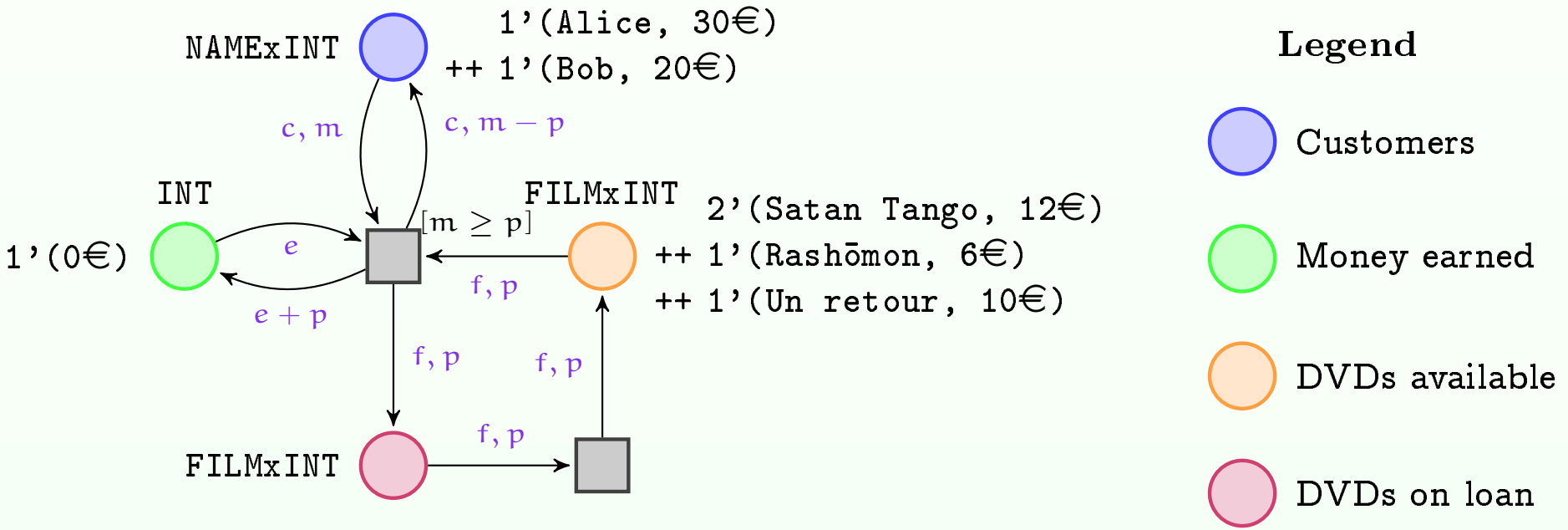
Colored Petri Nets [Jensen and Kristensen, 2009]

- Extension of Petri nets with colors
  - Tokens and places have a type ("color set")
  - Arcs are labeled with expressions
  - Transitions can have a guard

Colored Petri Nets [Jensen and Kristensen, 2009]

- Extension of Petri nets with colors
  - Tokens and places have a type ("color set")
  - Arcs are labeled with expressions
  - Transitions can have a guard

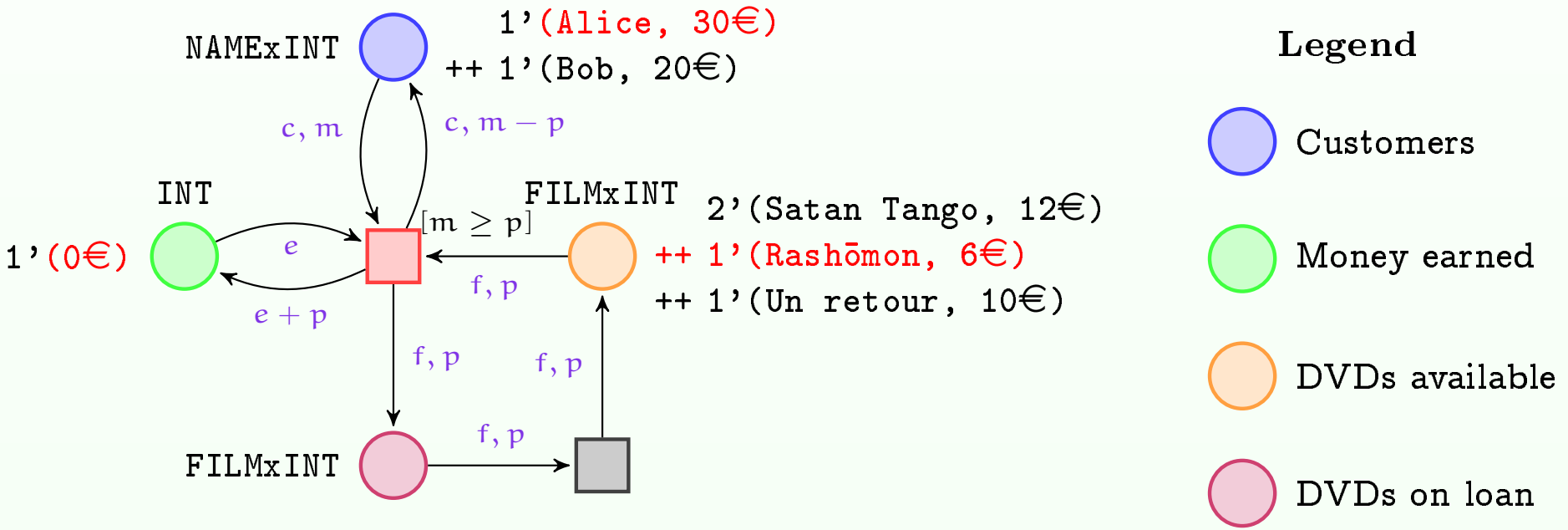
- Example: A more complex version of the DVD renting machine



Colored Petri Nets [Jensen and Kristensen, 2009]

- Extension of Petri nets with colors
  - Tokens and places have a type ("color set")
  - Arcs are labeled with expressions
  - Transitions can have a guard

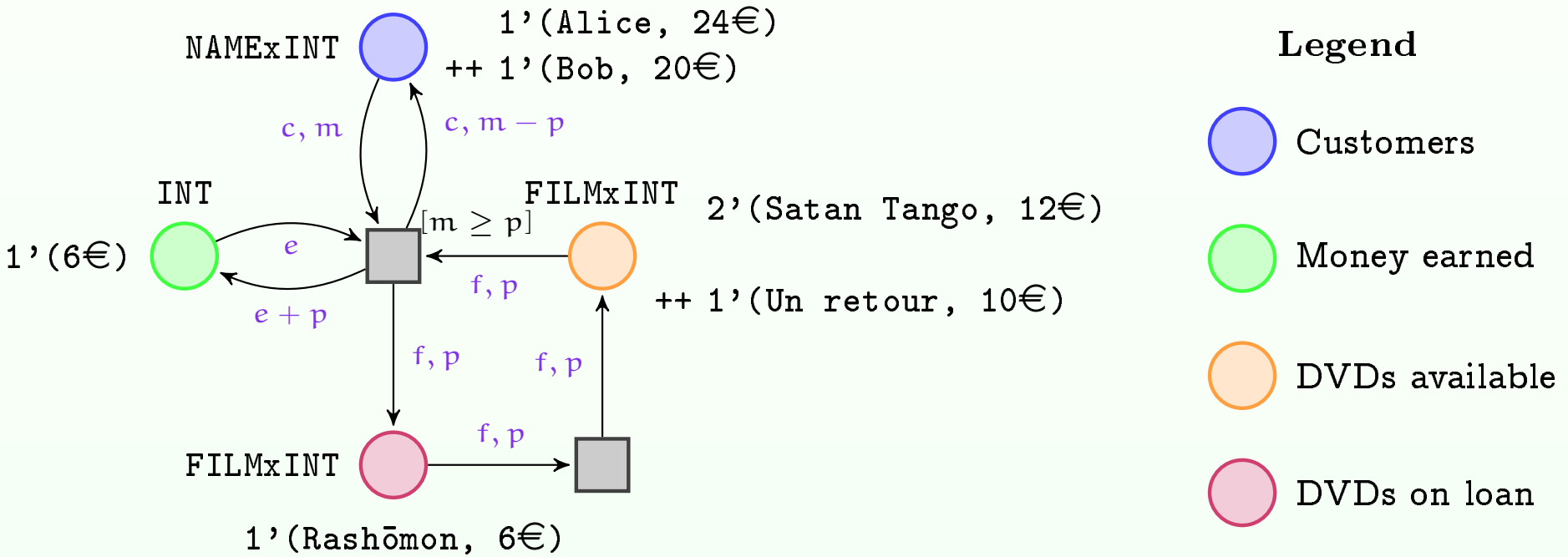
- Example: A more complex version of the DVD renting machine



Colored Petri Nets [Jensen and Kristensen, 2009]

- Extension of Petri nets with colors
  - Tokens and places have a type ("color set")
  - Arcs are labeled with expressions
  - Transitions can have a guard

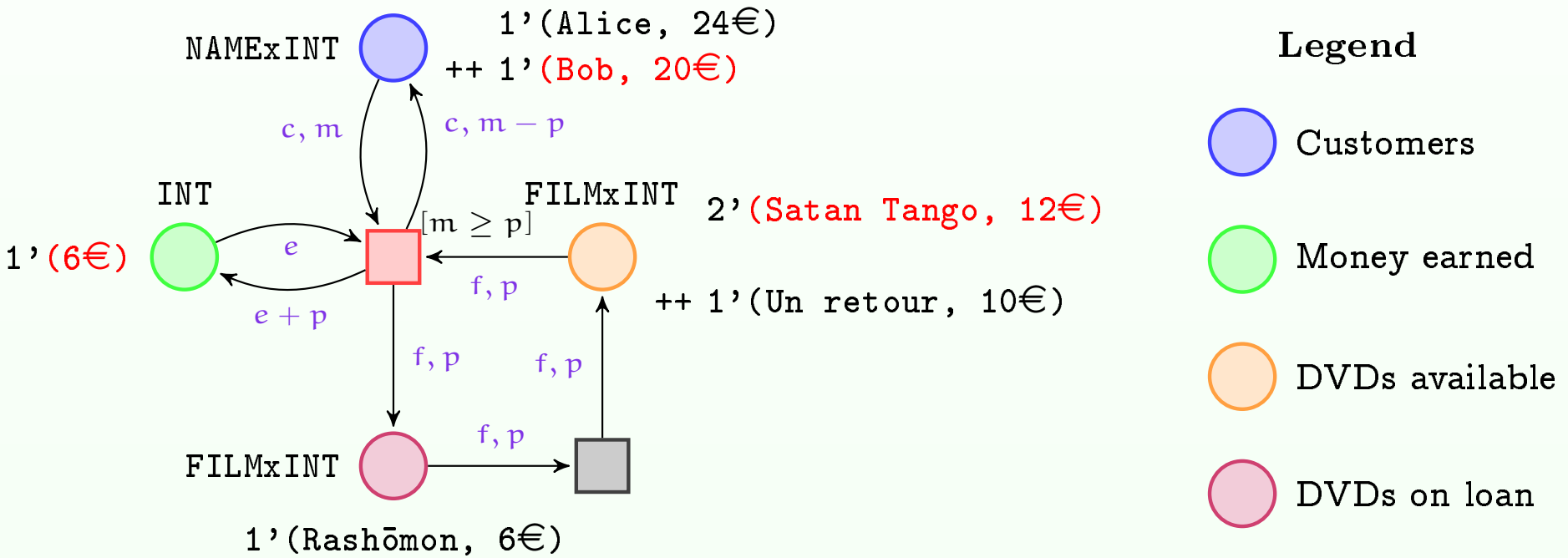
- Example: A more complex version of the DVD renting machine



Colored Petri Nets [Jensen and Kristensen, 2009]

- Extension of Petri nets with colors
  - Tokens and places have a type ("color set")
  - Arcs are labeled with expressions
  - Transitions can have a guard

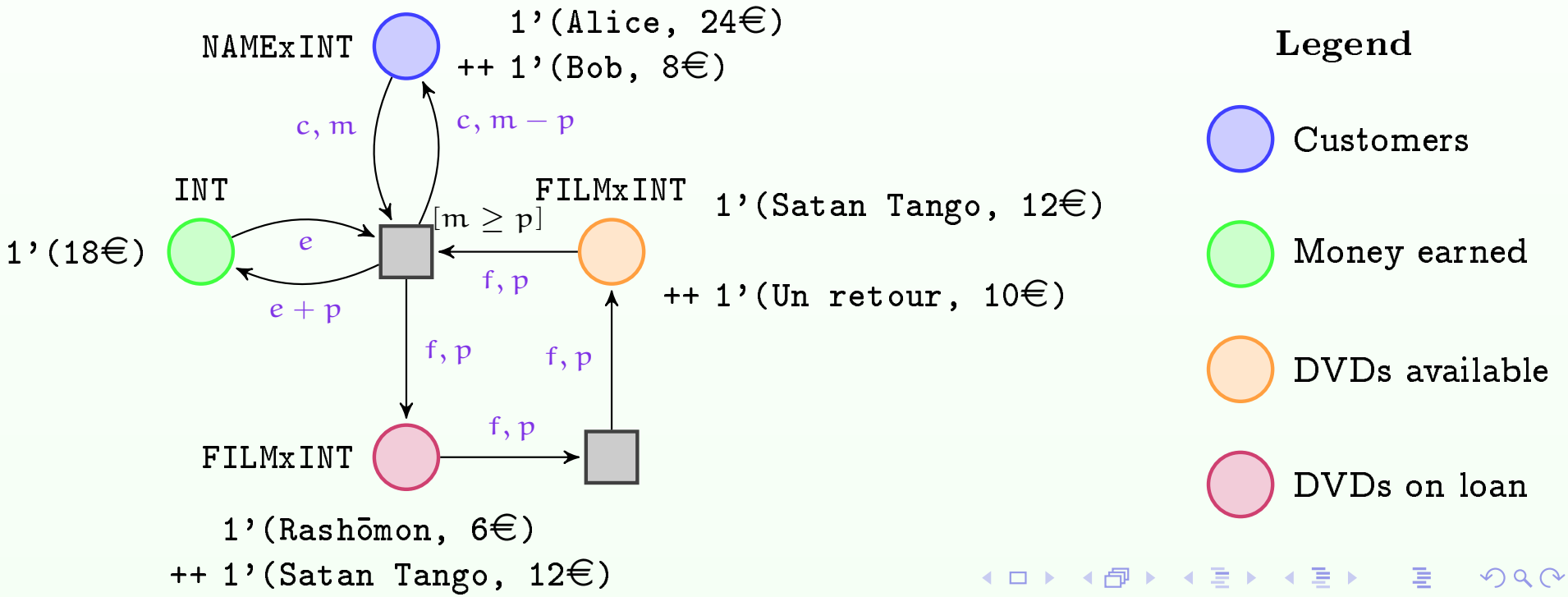
- Example: A more complex version of the DVD renting machine



Colored Petri Nets [Jensen and Kristensen, 2009]

- Extension of Petri nets with colors
  - Tokens and places have a type ("color set")
  - Arcs are labeled with expressions
  - Transitions can have a guard

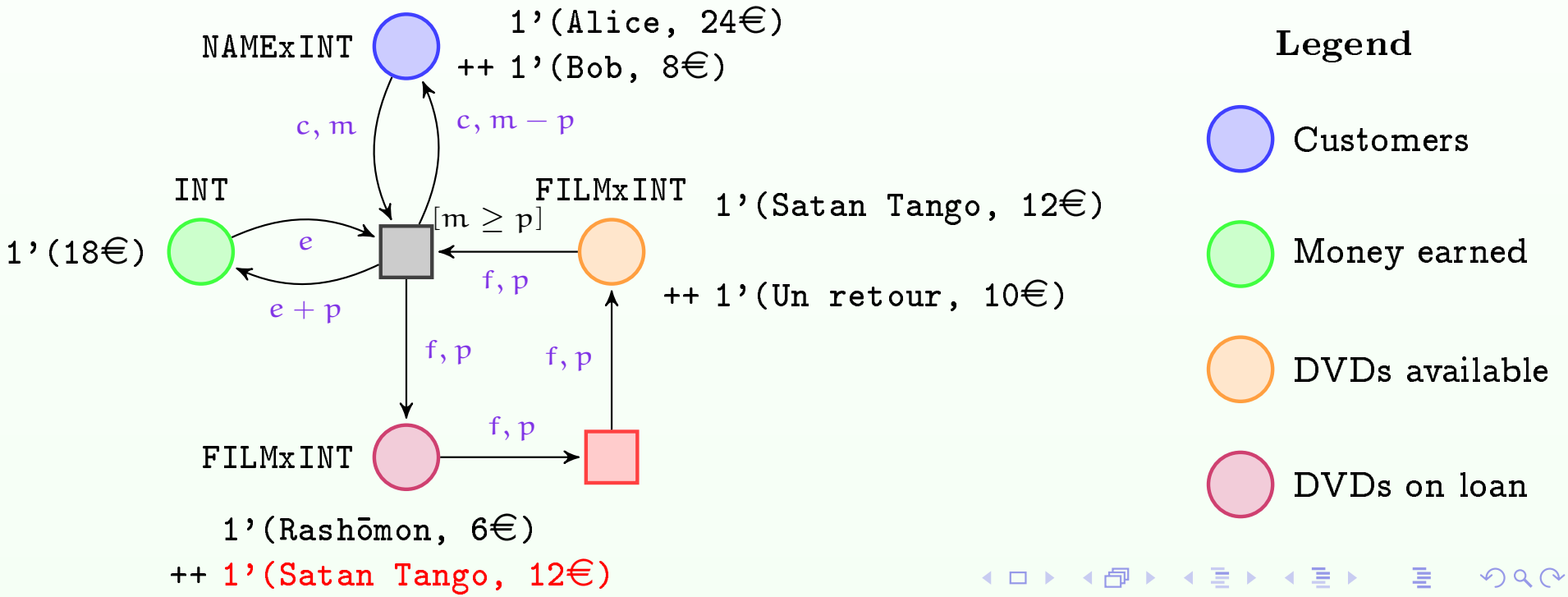
- Example: A more complex version of the DVD renting machine



Colored Petri Nets [Jensen and Kristensen, 2009]

- Extension of Petri nets with colors
  - Tokens and places have a type ("color set")
  - Arcs are labeled with expressions
  - Transitions can have a guard

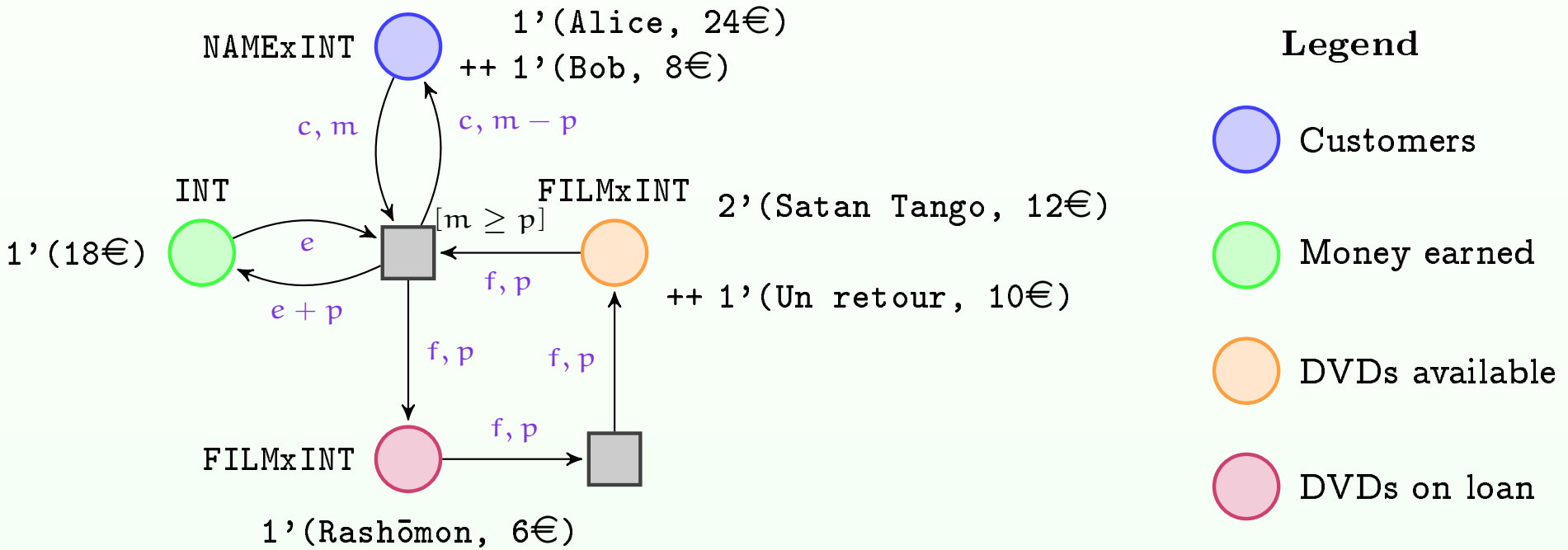
- Example: A more complex version of the DVD renting machine



Colored Petri Nets [Jensen and Kristensen, 2009]

- Extension of Petri nets with colors
  - Tokens and places have a type ("color set")
  - Arcs are labeled with expressions
  - Transitions can have a guard

- Example: A more complex version of the DVD renting machine



Outline

1 Colored Petri Nets

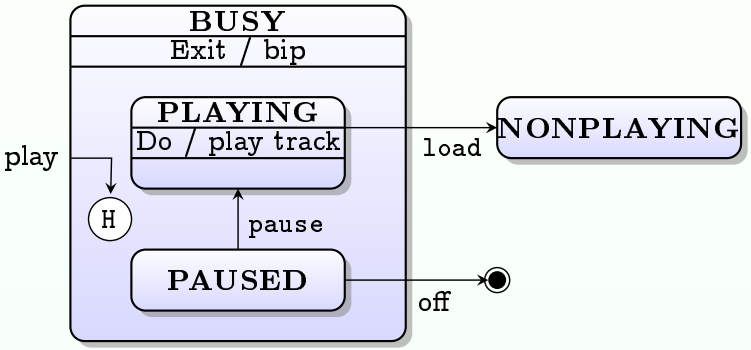
2 Translation

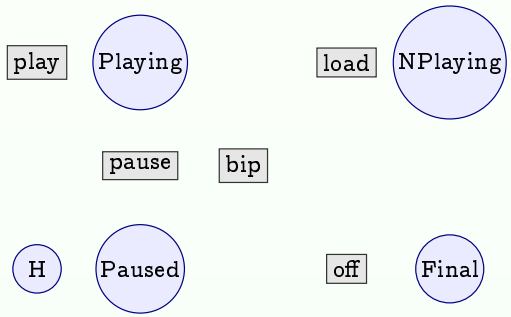
3 Application to the CD Player

4 Perspectives

Translating States and Behaviors

- Each simple state, final state, and history pseudostate is translated into a CPN place
  - Composite states not translated, only their simple substates are
- Each event and behavior is translated into a CPN transition





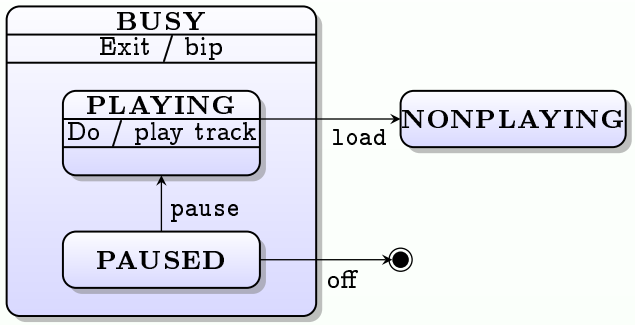
- Additional CPN places, transitions and arcs needed
  - Link between behaviors and states, etc.
  - Transitions

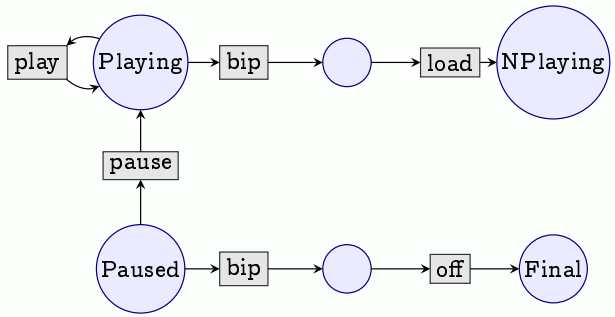
Typing

- Only one (colored) token in the resulting CPN
  - Possible because of non-concurrency
- This token carries:
  - The value of the global variables
  - The name of the latest substate for states containing a history pseudostate
- Example: CD Player
  - 2 global variables (present and track), one history pseudostate
  - Type of the token: $\{\texttt{true}, \texttt{false}\} \times \mathbb{N} \times \{\textbf{PLAYING}, \textbf{PAUSED}\}$
  - Example of value: $(\texttt{true}, 2, \textbf{PAUSED})$

Translating Transitions

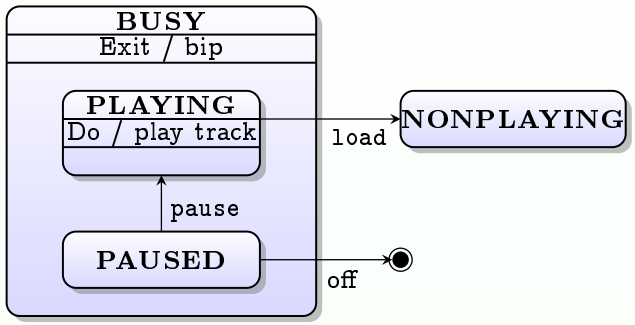
- A possible translation

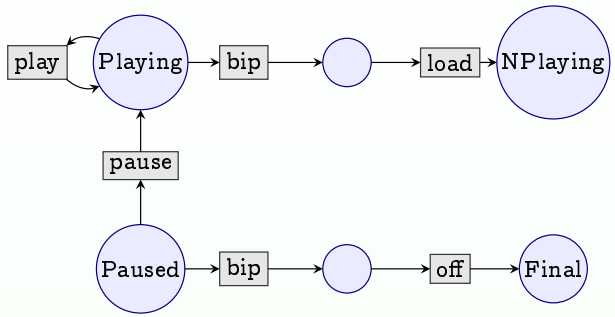




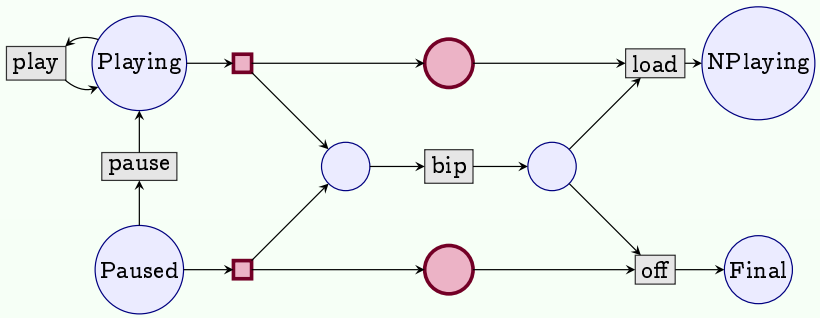
Translating Transitions

- A possible translation



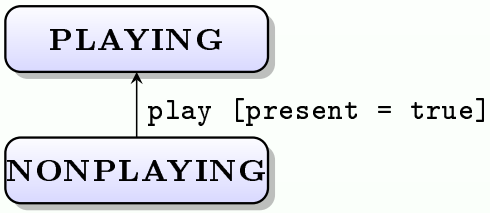


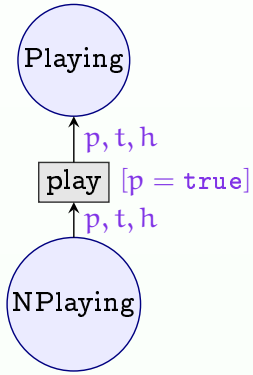
- Main problem: factoring
  - For each behavior in UML, we want only one CPN transition
  - Idea: use synchronization places



Translating Global Variables

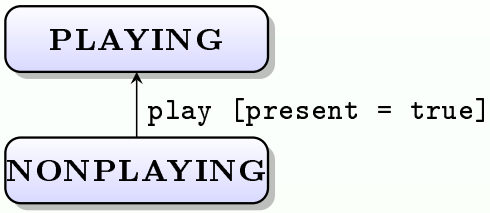
- Guards involving variables

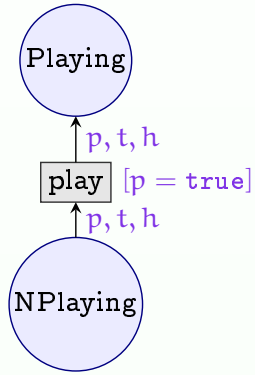




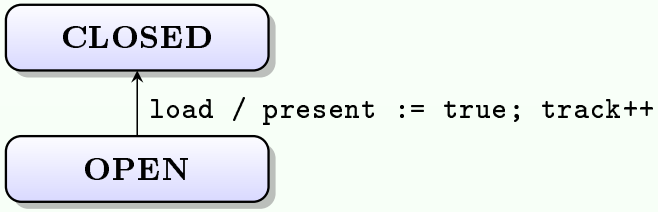
Translating Global Variables

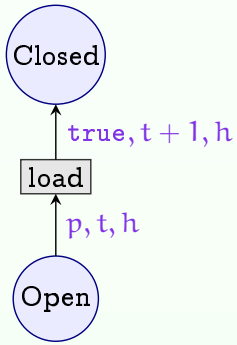
- Guards involving variables





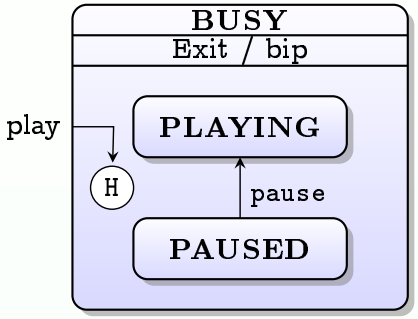
- Transitions updating variables

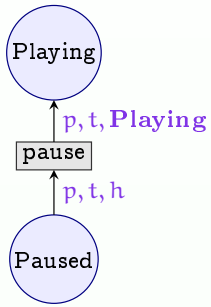




Translating History States

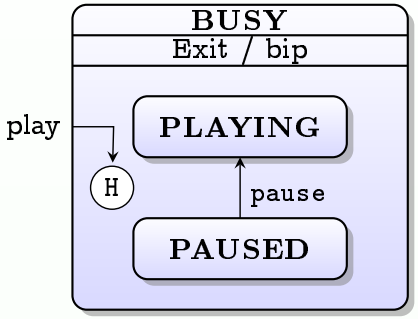
- For each transition leading to a substate, update the token

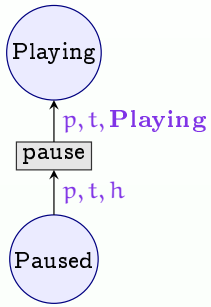




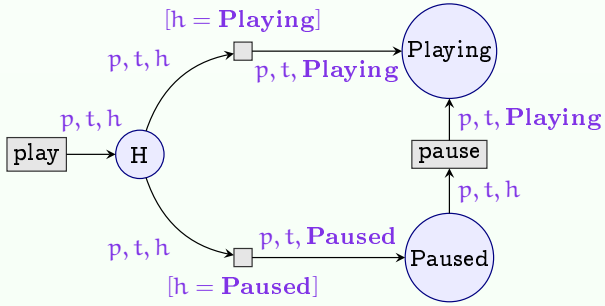
Translating History States

- For each transition leading to a substate, update the token





- For each history pseudostate, add CPN transitions to places corresponding to all substates, with the adequate guard



  - Note: this mechanism combines with the mechanism for behaviors (in case entry behaviors should be performed)

Outline

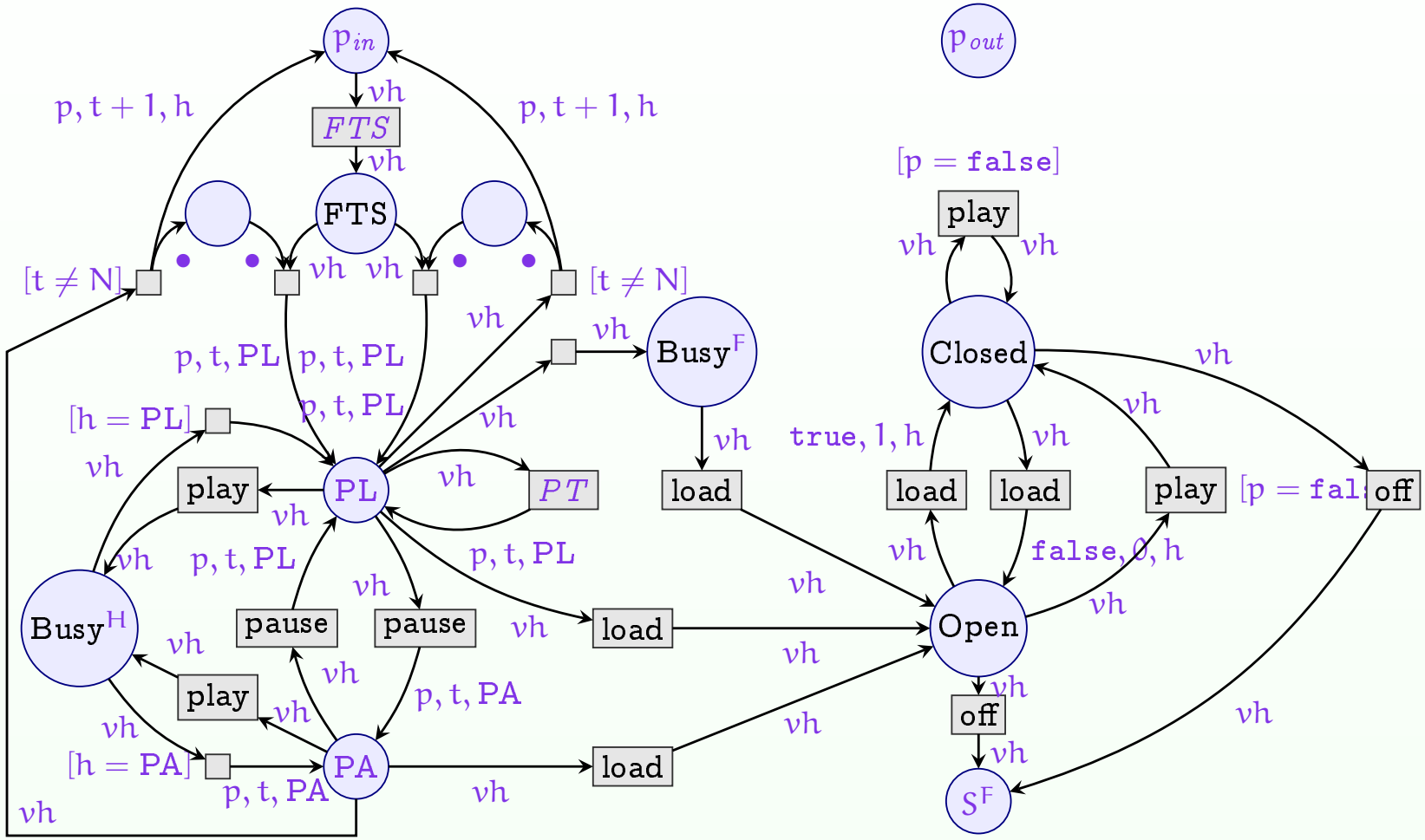
1 Colored Petri Nets

2 Translation

3 Application to the CD Player

4 Perspectives

Application to the CD Player (1/2)



where vh stands for p, t, h

Partial scheme

(See full scheme in [André et al., 2012])

Application to the CD Player (2/2)

- ☹ Scheme graphically more complex than the original diagram
  - But still possible to read visually
- ☺ But automated verification now possible

Application to the CD Player (2/2)

☹ Scheme graphically more complex than the original diagram

- But still possible to read visually

☺ But automated verification now possible

"When in **PLAYING**, there is a CD in the player"

Application to the CD Player (2/2)

☹ Scheme graphically more complex than the original diagram

- But still possible to read visually

☺ But automated verification now possible

√ "When in **PLAYING**, there is a CD in the player"

Application to the CD Player (2/2)

- ☹ Scheme graphically more complex than the original diagram
  - But still possible to read visually
- ☺ But automated verification now possible
  - √ "When in **PLAYING**, there is a CD in the player"
    "When in **PLAYING**, the track number is always between 1 and `trackCount`"

Application to the CD Player (2/2)

- ☹ Scheme graphically more complex than the original diagram
  - But still possible to read visually
- ☺ But automated verification now possible
  - √ "When in **PLAYING**, there is a CD in the player"
  - √ "When in **PLAYING**, the track number is always between 1 and `trackCount`"

Application to the CD Player (2/2)

☹ Scheme graphically more complex than the original diagram

- But still possible to read visually

☺ But automated verification now possible

√ "When in **PLAYING**, there is a CD in the player"

√ "When in **PLAYING**, the track number is always between 1 and `trackCount`"

(Properties verified using CPN Tools)

Outline

1 Colored Petri Nets

2 Translation

3 Application to the CD Player

4 Perspectives

Future Work

- Implementation and evaluation
- Extension to concurrent UML state machines
  - Use "global places" to encode global variables and history states
  - Use the mechanisms from [Choppy et al., 2011] to encode fork / join constructs
- Extension to full UML state machines 2.3 [OMG, 2010]
  - Including time issues
- Proof of equivalence with a semantics
  - Need to give a formal semantics to UML state machines first

References I

André, É., Choppy, C., and Klai, K. (2012).
Formalizing non-concurrent UML state machines using colored Petri nets (report).
Research report.
www-lipn.univ-paris13.fr/specif/documents/UMLPN.pdf.

Choppy, C., Klai, K., and Zidani, H. (2011).
Formal verification of UML state diagrams: a Petri net based approach.
SIGSOFT Softw. Eng. Notes, 36:1–8.

Clarke, E. M. and Heinle, W. (2000).
Modular translation of statecharts to SMV.
Technical report, Carnegie Mellon University School of Computer Science.

Dubrovin, J. and Junttila, T. A. (2007).
Symbolic model checking of hierarchical UML state machines.
Technical Report B23, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland.

Jensen, K. and Kristensen, L. M. (2009).
Coloured Petri Nets – Modelling and Validation of Concurrent Systems.
Springer.

References II

Jin, Y., Esser, R., and Janneck, J. W. (2004).
A method for describing the syntax and semantics of UML statecharts.
Software and System Modeling, 3(2):150–163.

Latella, D., Majzik, I., and Massink, M. (1999).
Automatic verification of a behavioural subset of UML statechart diagrams using the SPIN model-checker.
Formal Aspects of Computing, V11(6):637–664.

Lian, J., Hu, Z., and Shatz, S. M. (2008).
Simulation-based analysis of UML statechart diagrams: methods and case studies.
Software Quality Journal, 16(1):45–78.

OMG (2009).
Unified Modeling Language Superstructure Version 2.2.
http://www.omg.org/spec/UML/2.2/Superstructure/PDF/.

OMG (2010).
OMG Unified Modeling Language Superstructure, Version 2.3.
http://www.omg.org/spec/UML/2.3/Superstructure/PDF/.

References III


Petri, C. A. (1962).
Kommunikation mit Automaten.
PhD thesis, Institut für Instrumentelle Mathematik.


Pettit IV, R. G. and Gomaa, H. (2006).
Modeling behavioral patterns of concurrent objects using Petri nets.
In *ISORC*, pages 303–312.


Zhang, S. and Liu, Y. (2010).
An automatic approach to model checking UML state machines.
In *SSIRI-C'10*, pages 1–6. IEEE.