Petri nets, behavioural semantics and web services development

F. Gadducci

Pisa University

[joint work with F. Bonchi, A. Brogi, and S.Corfini]

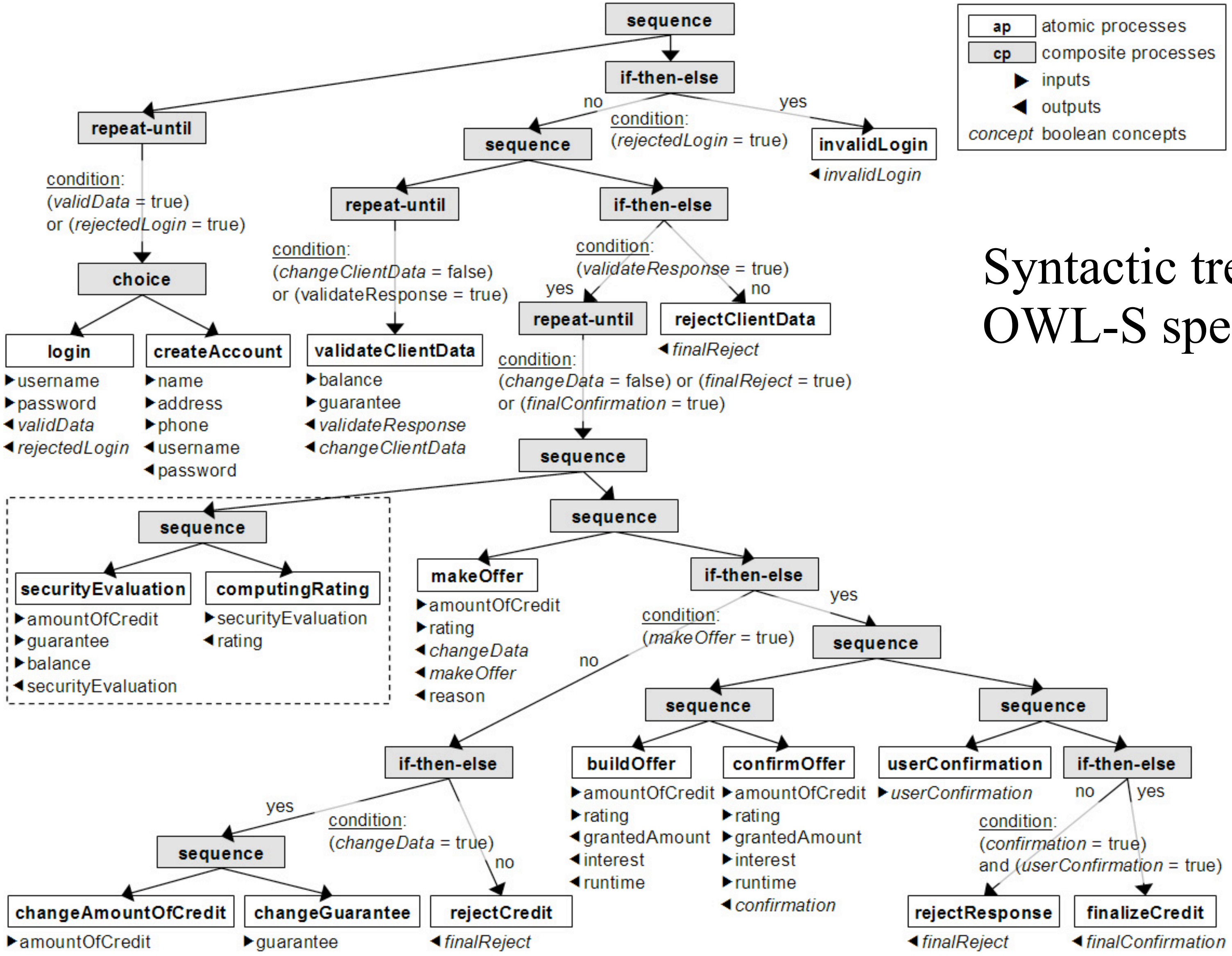
Plan of the talk

- From Web Services to CPR nets
- Open CPR nets and Contexts
- Saturated Semantics for CPR nets
- Conclusions (and example)

Plan of the talk

- **From Web Services to CPR nets**
- Open CPR nets and Contexts
- Saturated Semantics for CPR nets
- Conclusions (and example)

Syntactic tree of an OWL-S specification



ap atomic processes
cp composite processes
▶ inputs
◀ outputs
concept boolean concepts

sequence

if-then-else

no

yes

condition: (*rejectedLogin* = true)

invalidLogin
◀ *invalidLogin*

repeat-until

condition: (*validData* = true) or (*rejectedLogin* = true)

choice

login
▶username
▶password
◀*validData*
◀*rejectedLogin*

createAccount
▶name
▶address
▶phone
◀username
◀password

sequence

repeat-until

condition: (*changeClientData* = false) or (validateResponse = true)

validateClientData
▶balance
▶guarantee
◀*validateResponse*
◀*changeClientData*

if-then-else

condition: (*validateResponse* = true)

yes

no

rejectClientData
◀*finalReject*

repeat-until

condition: (*changeData* = false) or (*finalReject* = true) or (*finalConfirmation* = true)

sequence

sequence

securityEvaluation
▶amountOfCredit
▶guarantee
▶balance
◀securityEvaluation

computingRating
▶securityEvaluation
◀rating

sequence

makeOffer
▶amountOfCredit
▶rating
◀*changeData*
◀*makeOffer*
◀reason

if-then-else

condition: (*makeOffer* = true)

yes

no

sequence

sequence

buildOffer
▶amountOfCredit
▶rating
◀grantedAmount
◀interest
◀runtime

confirmOffer
▶amountOfCredit
▶rating
▶grantedAmount
▶interest
▶runtime
◀*confirmation*

sequence

userConfirmation
▶*userConfirmation*

if-then-else

no

yes

condition: (*confirmation* = true) and (*userConfirmation* = true)

rejectResponse
◀*finalReject*

finalizeCredit
◀*finalConfirmation*

if-then-else

yes

condition: (*changeData* = true)

no

sequence

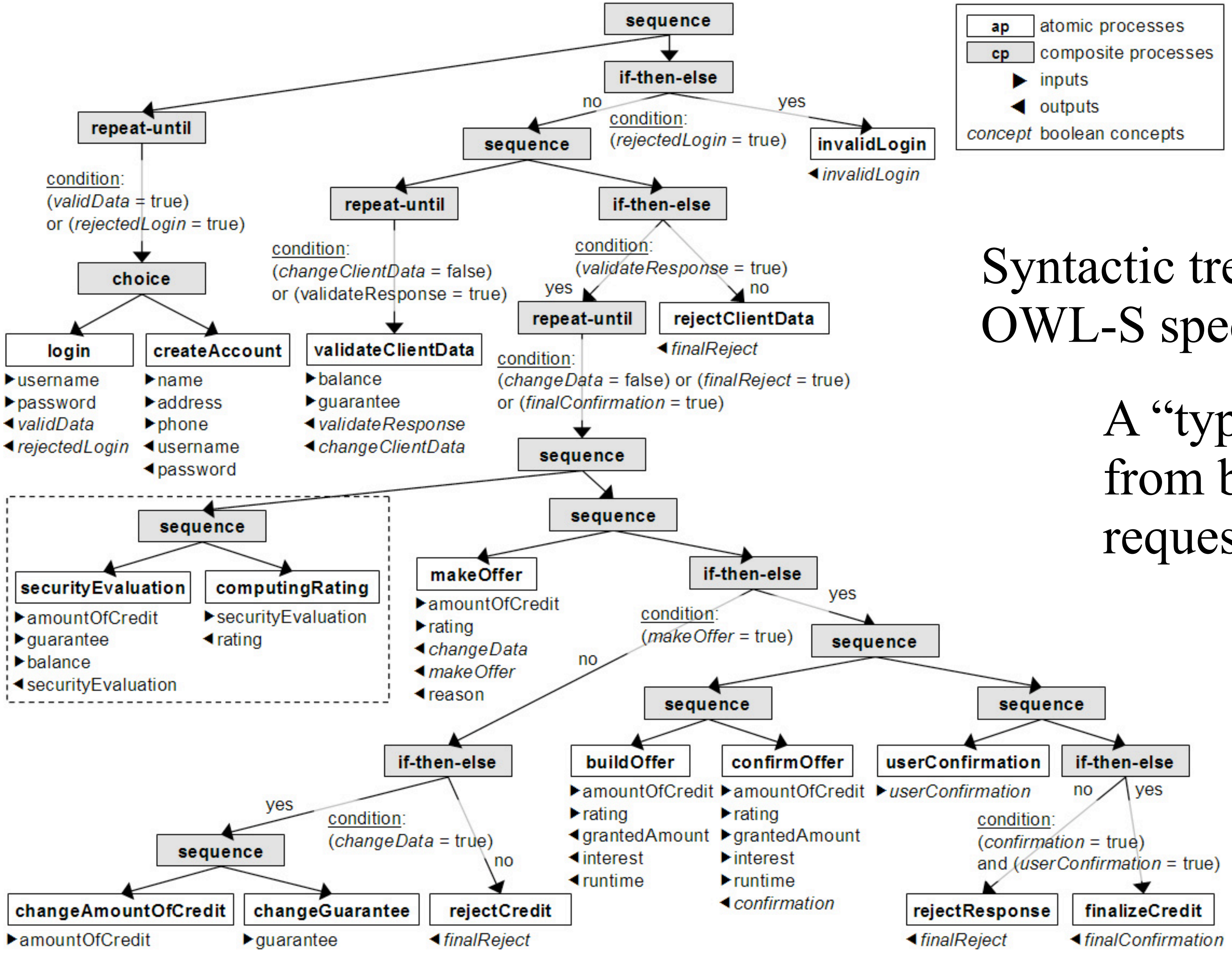
changeAmountOfCredit
▶amountOfCredit

changeGuarantee
▶guarantee

rejectCredit
◀*finalReject*

Syntactic tree of an OWL-S specification

A “typical” example from banking: credit request procedure



ap atomic processes
cp composite processes
▶ inputs
◀ outputs
concept boolean concepts

sequence

if-then-else
no
yes
condition:
(*rejectedLogin* = true)

invalidLogin
◀ *invalidLogin*

repeat-until
condition:
(*validData* = true)
or (*rejectedLogin* = true)

choice

login
▶username
▶password
◀ *validData*
◀ *rejectedLogin*

createAccount
▶name
▶address
▶phone
◀username
◀password

sequence

repeat-until
condition:
(*changeClientData* = false)
or (validateResponse = true)

validateClientData
▶balance
▶guarantee
◀ *validateResponse*
◀ *changeClientData*

if-then-else
condition:
(*validateResponse* = true)
yes
no

rejectClientData
◀ *finalReject*

repeat-until
condition:
(*changeData* = false) or (*finalReject* = true)
or (*finalConfirmation* = true)

sequence

sequence

securityEvaluation
▶amountOfCredit
▶guarantee
▶balance
◀securityEvaluation

computingRating
▶securityEvaluation
◀rating

sequence

makeOffer
▶amountOfCredit
▶rating
◀ *changeData*
◀ *makeOffer*
◀reason

if-then-else
yes
condition:
(*makeOffer* = true)
no

if-then-else
yes
condition:
(*changeData* = true)
no

sequence

changeAmountOfCredit
▶amountOfCredit

changeGuarantee
▶guarantee

rejectCredit
◀ *finalReject*

sequence

sequence

buildOffer
▶amountOfCredit
▶rating
◀grantedAmount
◀interest
◀runtime

confirmOffer
▶amountOfCredit
▶rating
▶grantedAmount
▶interest
▶runtime
◀ *confirmation*

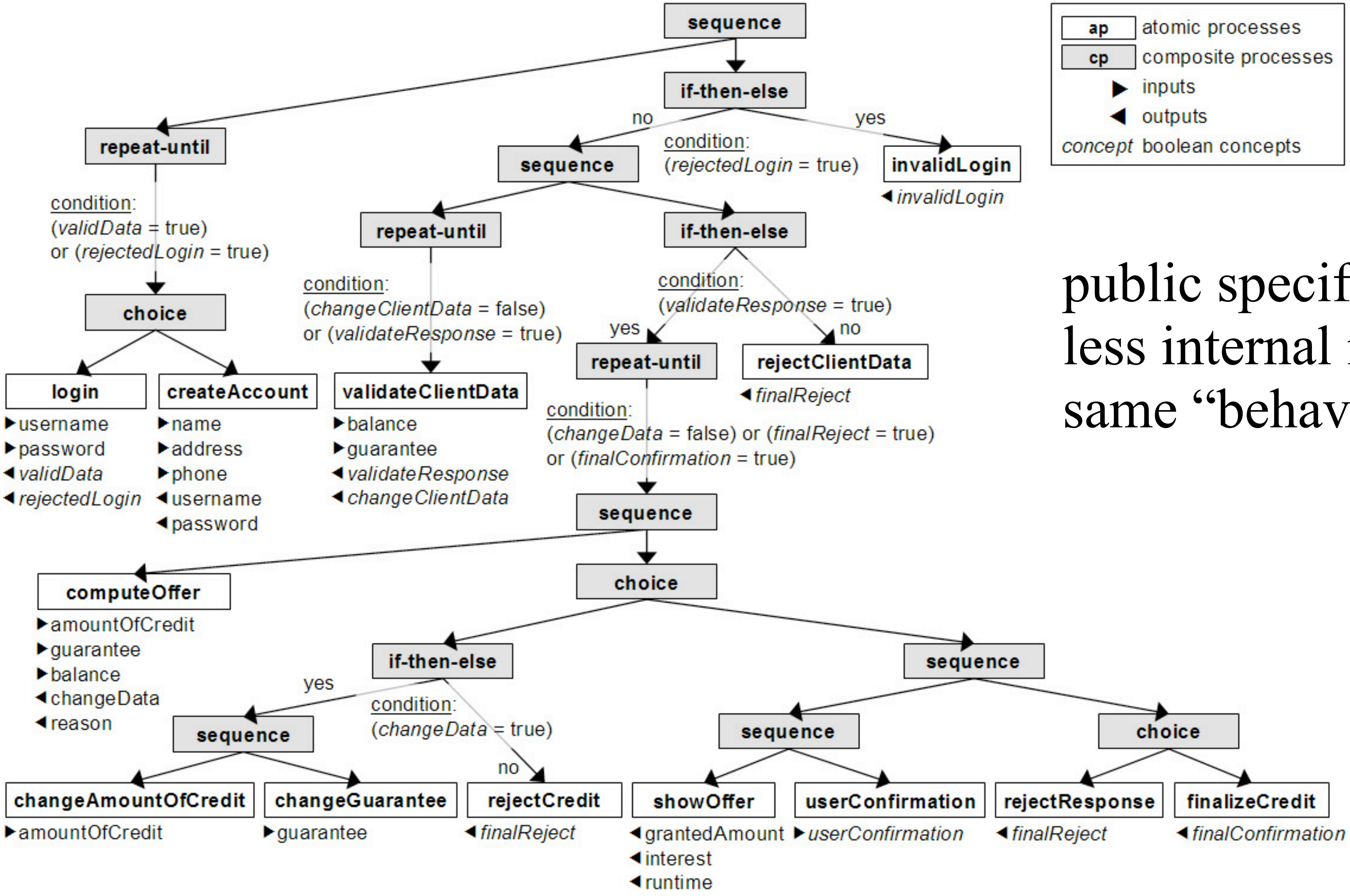
sequence

userConfirmation
▶ *userConfirmation*

if-then-else
no
yes
condition:
(*confirmation* = true)
and (*userConfirmation* = true)

rejectResponse
◀ *finalReject*

finalizeCredit
◀ *finalConfirmation*



ap atomic processes
cp composite processes
▶ inputs
◀ outputs
concept boolean concepts

sequence

repeat-until

condition:
(*validData* = true)
or (*rejectedLogin* = true)

choice

login
▶username
▶password
◀*validData*
◀*rejectedLogin*

createAccount
▶name
▶address
▶phone
◀username
◀password

if-then-else

no

yes

condition:
(*rejectedLogin* = true)

invalidLogin
◀*invalidLogin*

sequence

repeat-until

condition:
(*changeClientData* = false)
or (*validateResponse* = true)

validateClientData
▶balance
▶guarantee
◀*validateResponse*
◀*changeClientData*

if-then-else

condition:
(*validateResponse* = true)

yes

no

rejectClientData
◀*finalReject*

repeat-until

condition:
(*changeData* = false) or (*finalReject* = true)
or (*finalConfirmation* = true)

sequence

computeOffer
▶amountOfCredit
▶guarantee
▶balance
◀changeData
◀reason

choice

if-then-else

yes

condition:
(*changeData* = true)

no

sequence

changeAmountOfCredit
▶amountOfCredit

changeGuarantee
▶guarantee

rejectCredit
◀*finalReject*

sequence

sequence

showOffer
◀grantedAmount
◀interest
◀runtime

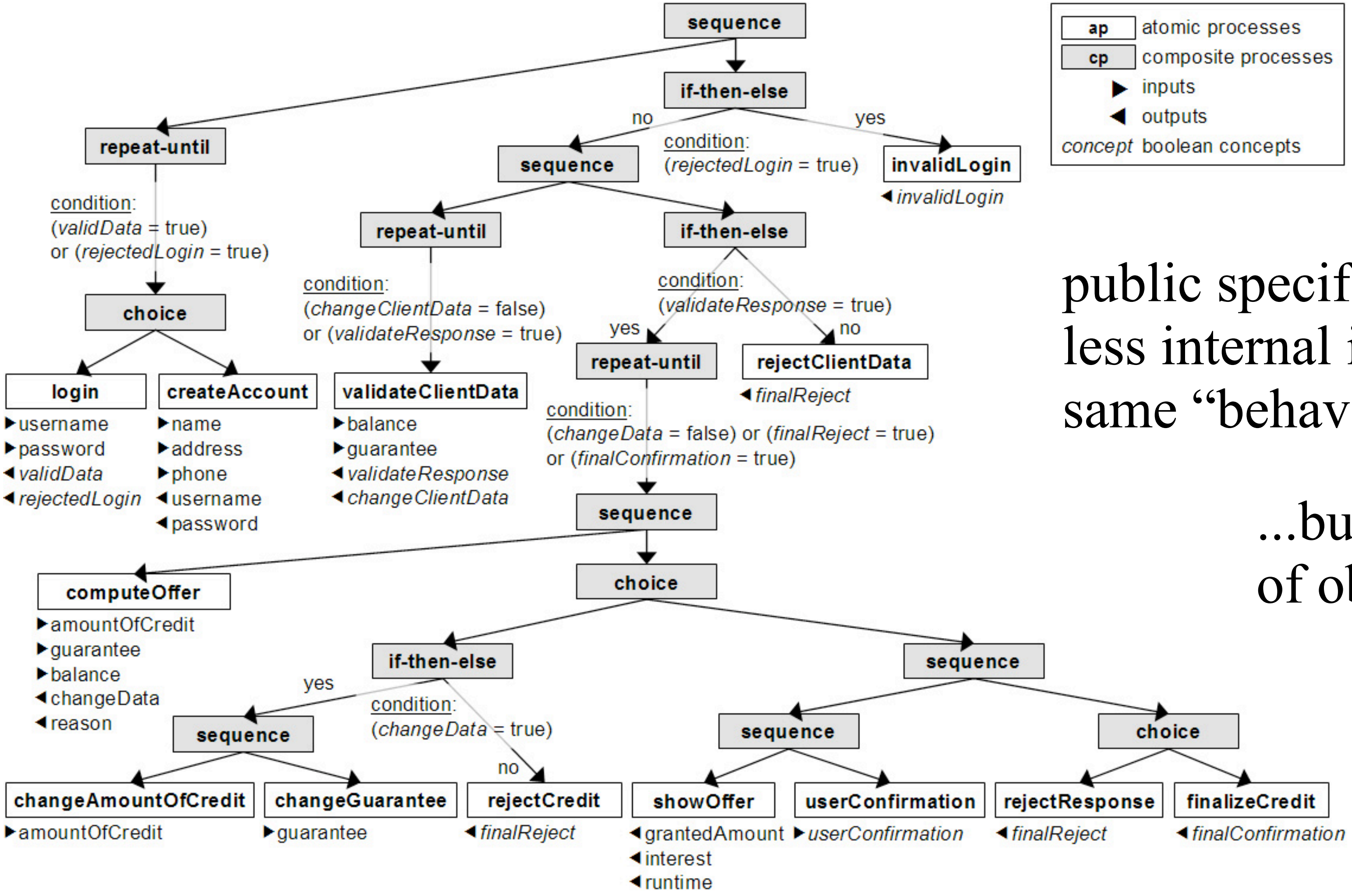
userConfirmation
▶*userConfirmation*

choice

rejectResponse
◀*finalReject*

finalizeCredit
◀*finalConfirmation*

public specification:
less internal info, yet
same “behaviour”...



public specification:
less internal info, yet
same “behaviour”...

...but which kind
of observations?

From Web Services to Petri nets

B. Benatallah and R. Hamadi. A Petri net-based model for Web service composition. In K.-D. Schewe and X. Zhou, editors, *Australasian Database Conference, Conferences in Research and Practice in Information Technology* 17, pp. 191–200. Australian Computer Society, 2003.

A. Martens. Analyzing Web service based business processes. In M. Cerioli, editor, *Fundamental Approaches to Software Engineering, LNCS* 3442, pp. 19–33. Springer, 2005.

A. Martens. Consistency between executable and abstract processes. In *e-Technology, e-Commerce, and e-Services*, pp. 60–67. IEEE Computer Society, 2005.

H.M.W Verbeek and W.M.P. van der Aalst. Analyzing BPEL processes using Petri nets. In D. Marinescu, editor, *Applications of Petri Nets to Coordination, Workflow and Business Process Management*, pp. 59–78. Florida International University, 2005.

Composition of Web Services

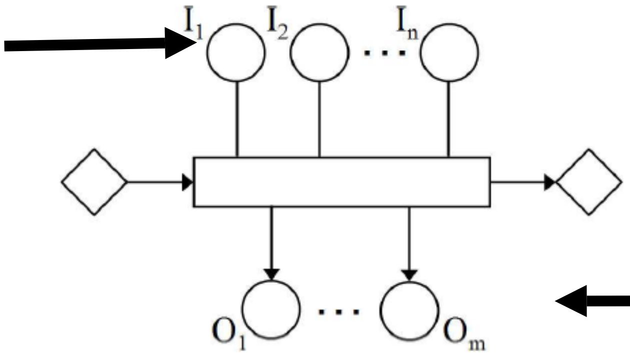
A. Brogi, S. Corfini. **Behaviour-aware discovery of Web service compositions**

International Journal of Web Service Research 4(3), pp. 1-25, 2007.

- **OWL-S** ontologies (proposed by W3C)
- An OWL-S advertisement contains three descriptions:
  - *Service Profile*: “what the service does”
  - ***Service Model***: “how the service works”
  - *Service Grounding*: “how to access the service”

From OWL-S to Petri nets

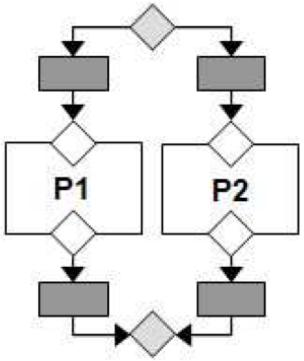
Input data



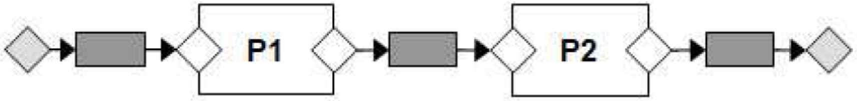
Output data

Atomic Process

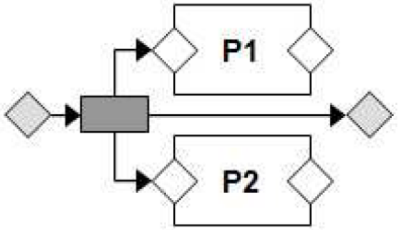
From OWL-S to Petri nets



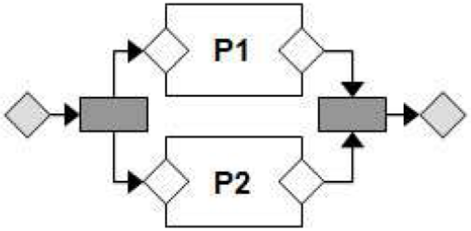
Choice(P1,P2)



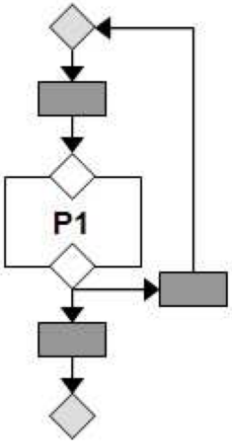
Sequence(P1,P2)



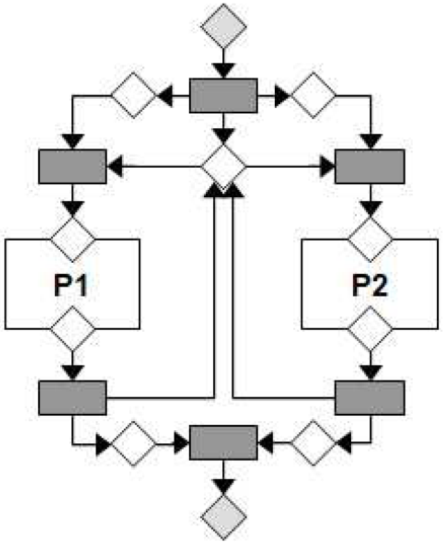
Split(P1,P2)



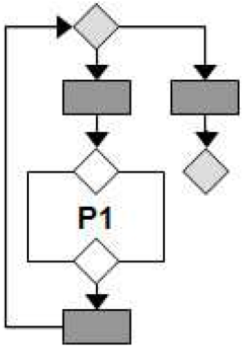
Split+Join(P1,P2)



Repeat-Until(P1)

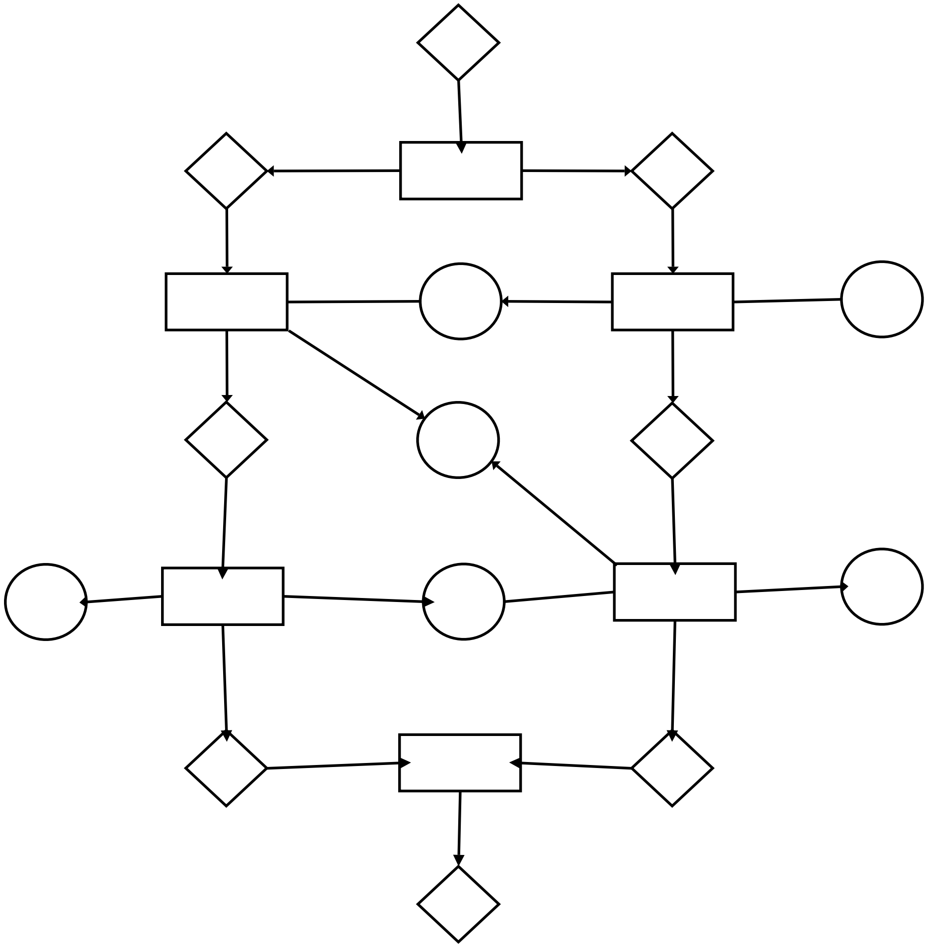


Any-order(P1,P2)

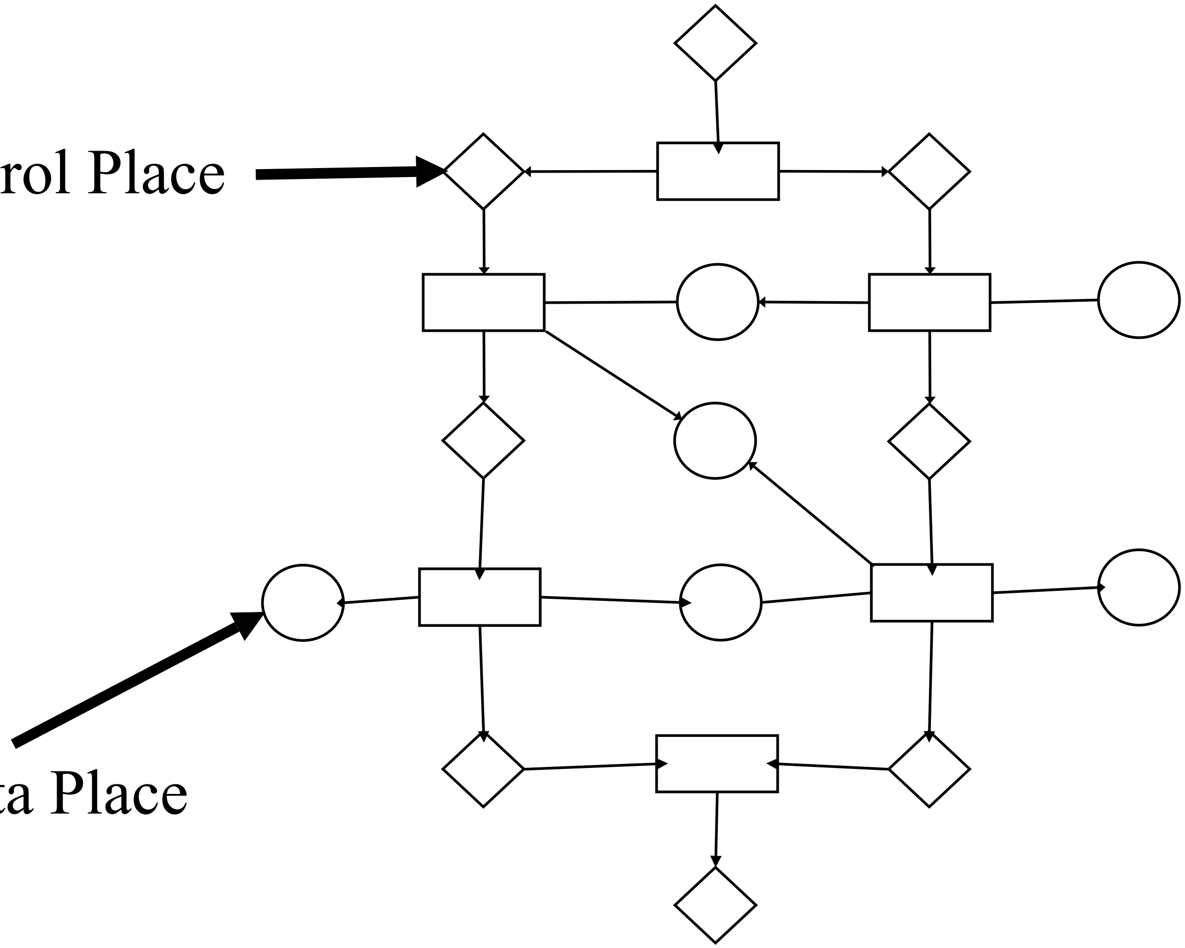


Repeat-While(P1)

Consume-Produce-Read nets



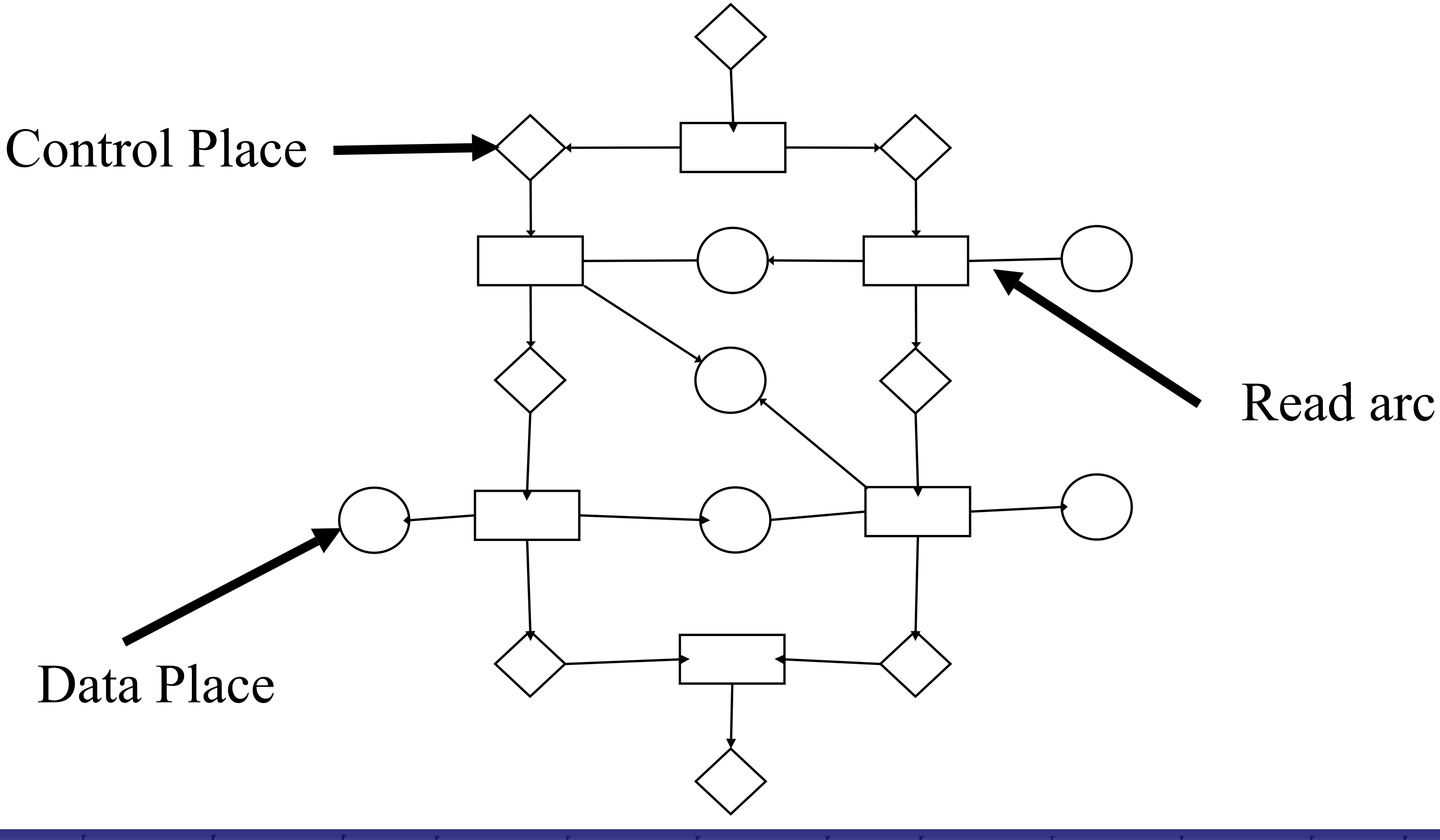
Consume-Produce-Read nets



Control Place

Data Place

Consume-Produce-Read nets



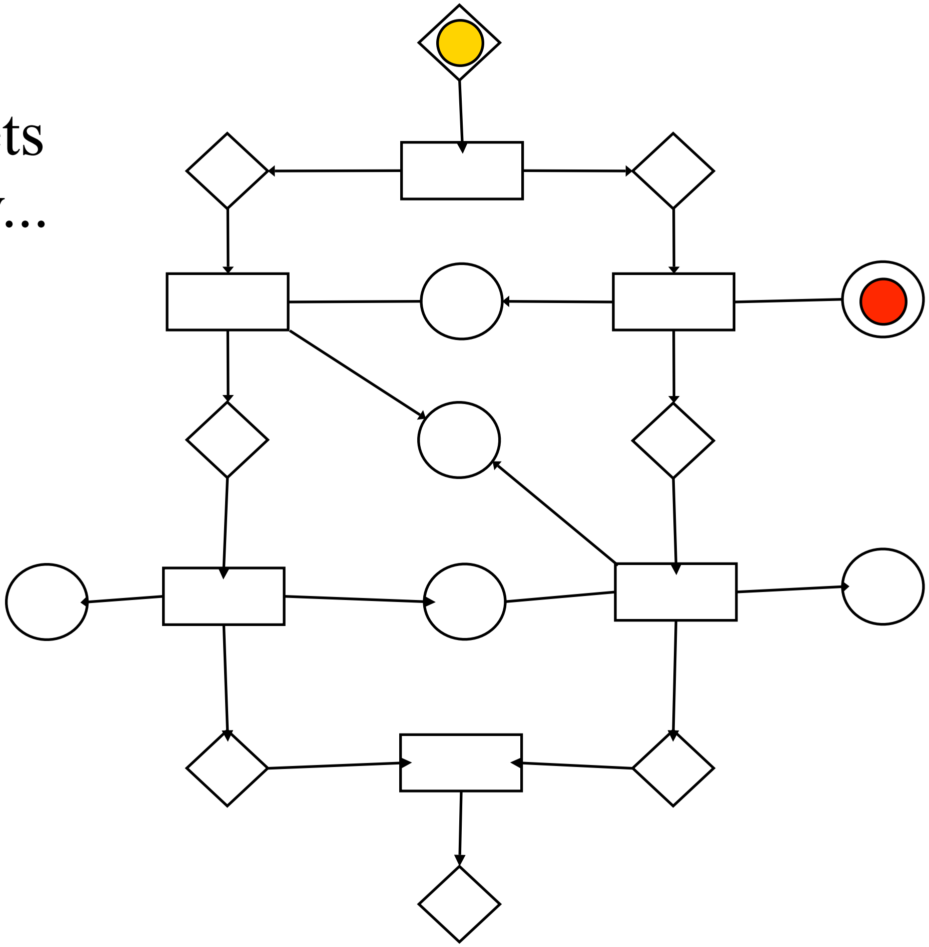
Control Place

Read arc

Data Place

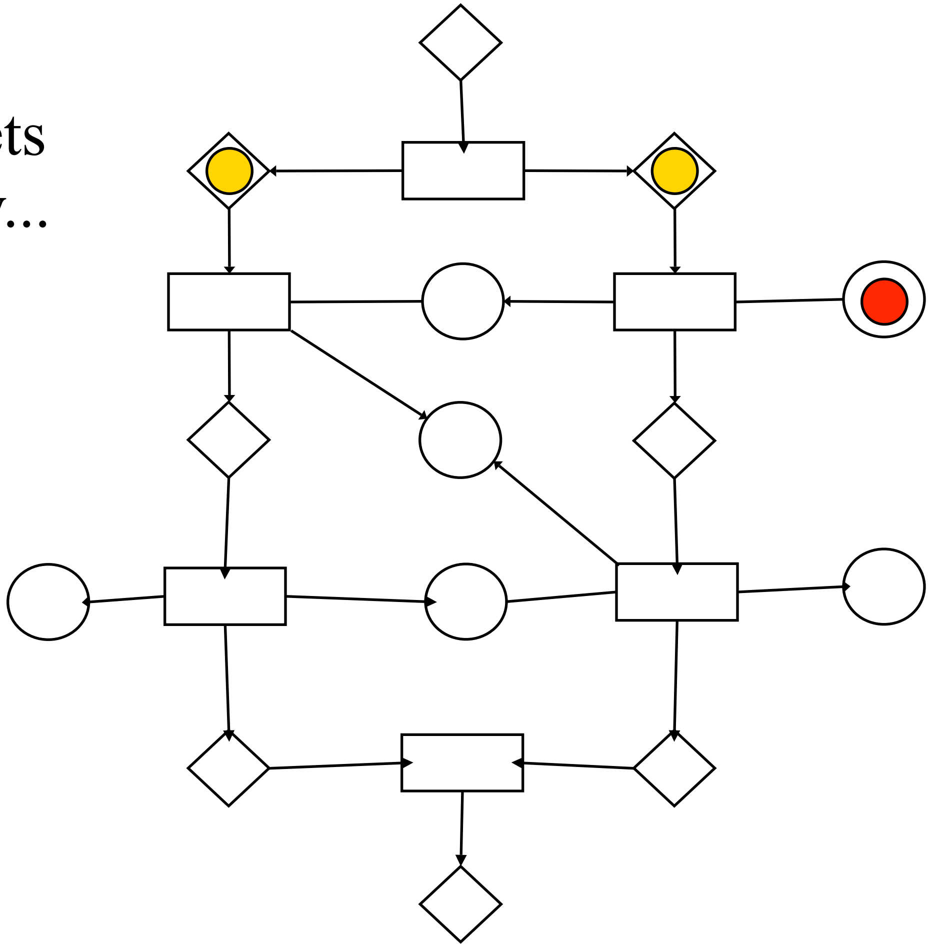
The token game

Behave as C/E nets w.r.t. control flow...



The token game

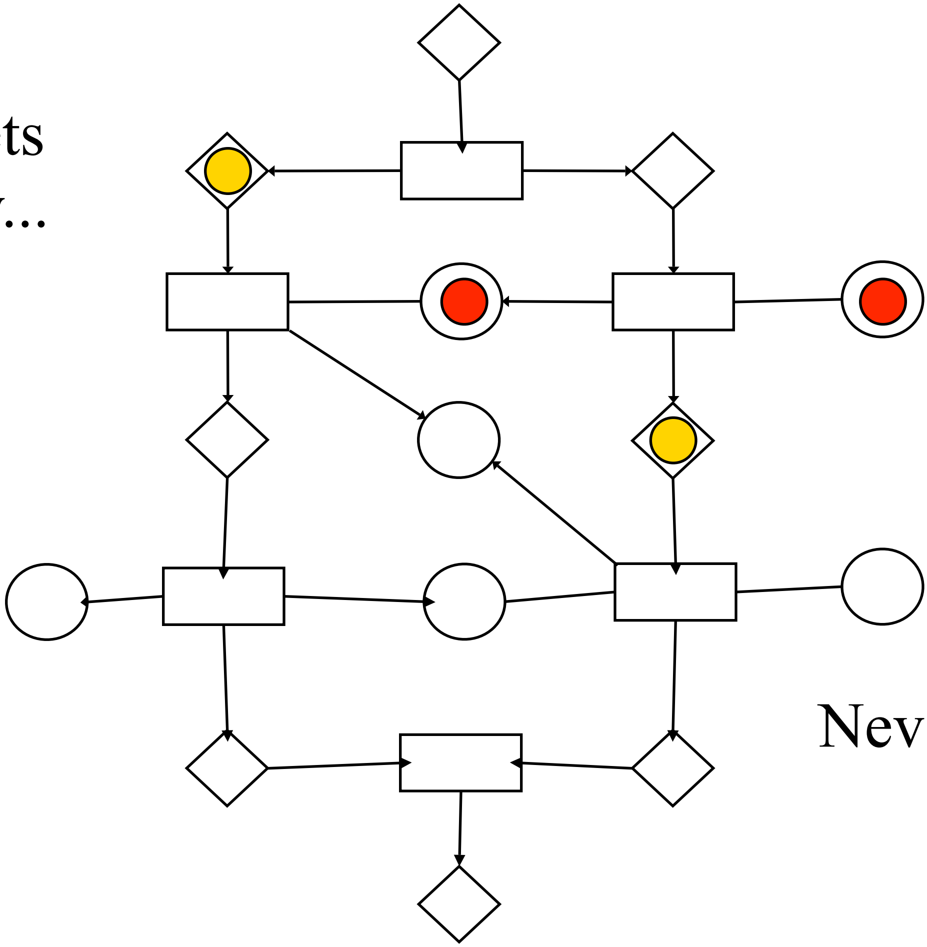
Behave as C/E nets w.r.t. control flow...



The token game

Behave as C/E nets
w.r.t. control flow...

but data places
are persistent!

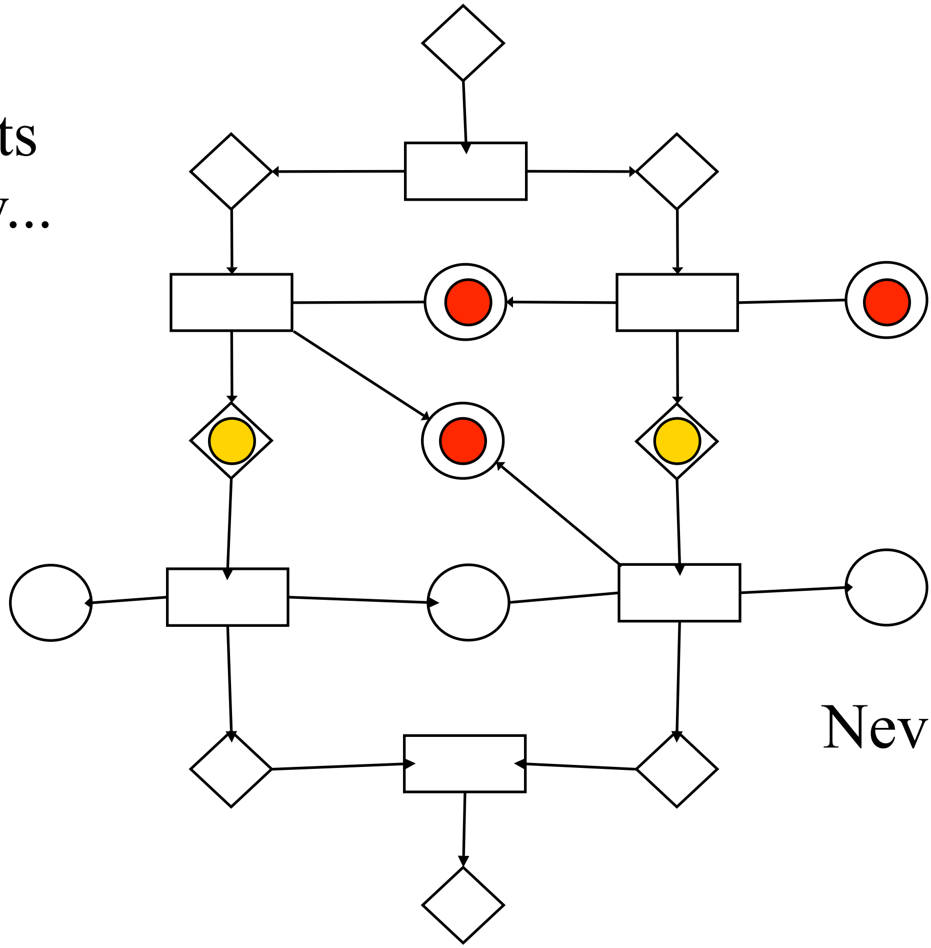


Never consumed

The token game

behave as C/E nets
w.r.t. control flow...

but data places
are persistent!

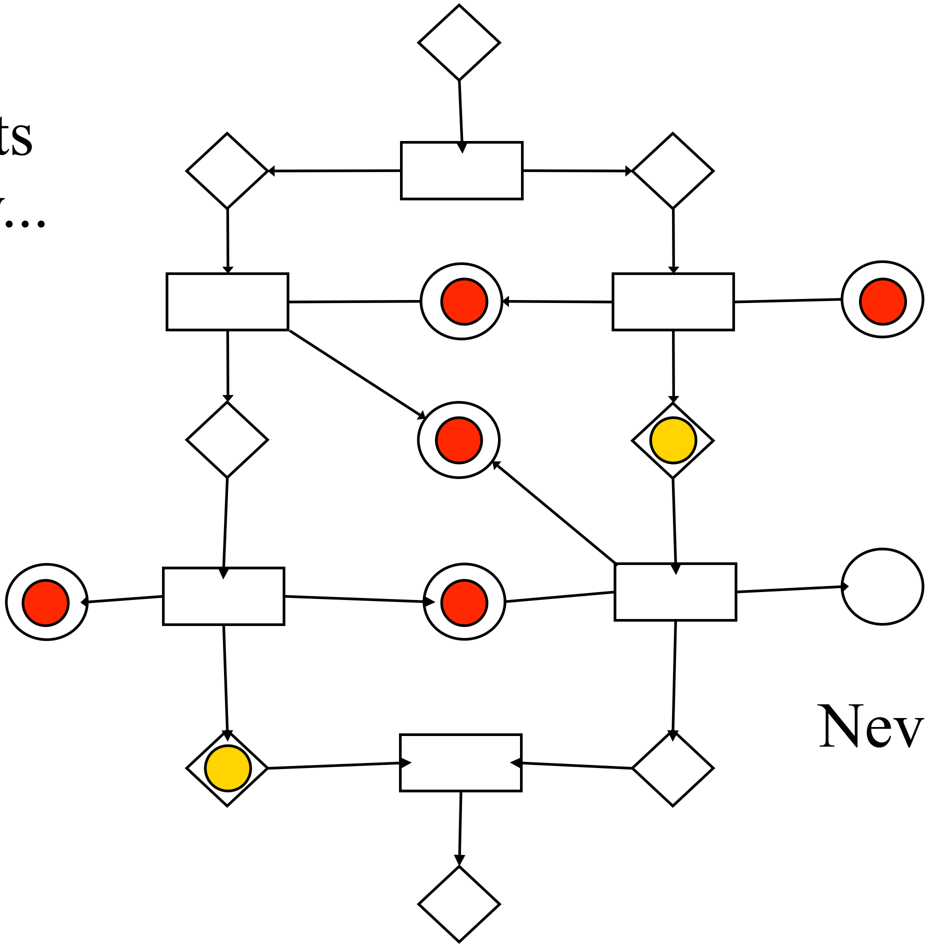


Never consumed

The token game

behave as C/E nets
w.r.t. control flow...

but data places
are persistent!

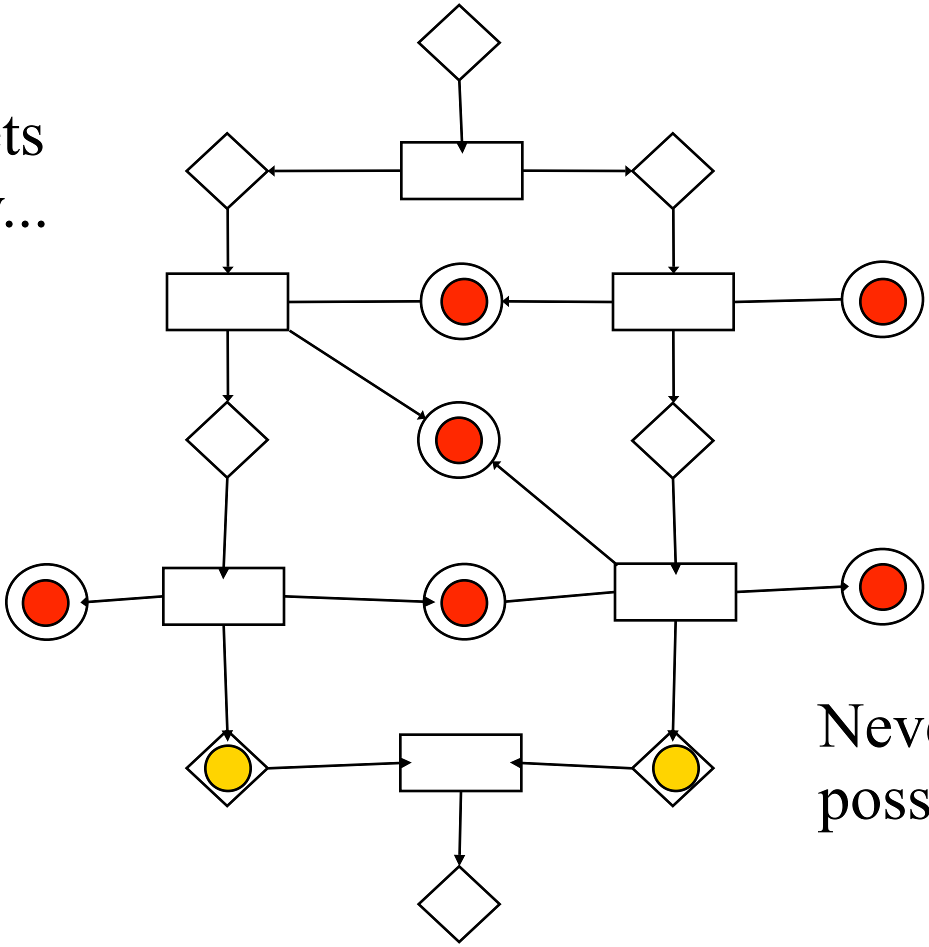


Never consumed

The token game

Behave as C/E nets
w.r.t. control flow...

but data places
are persistent!

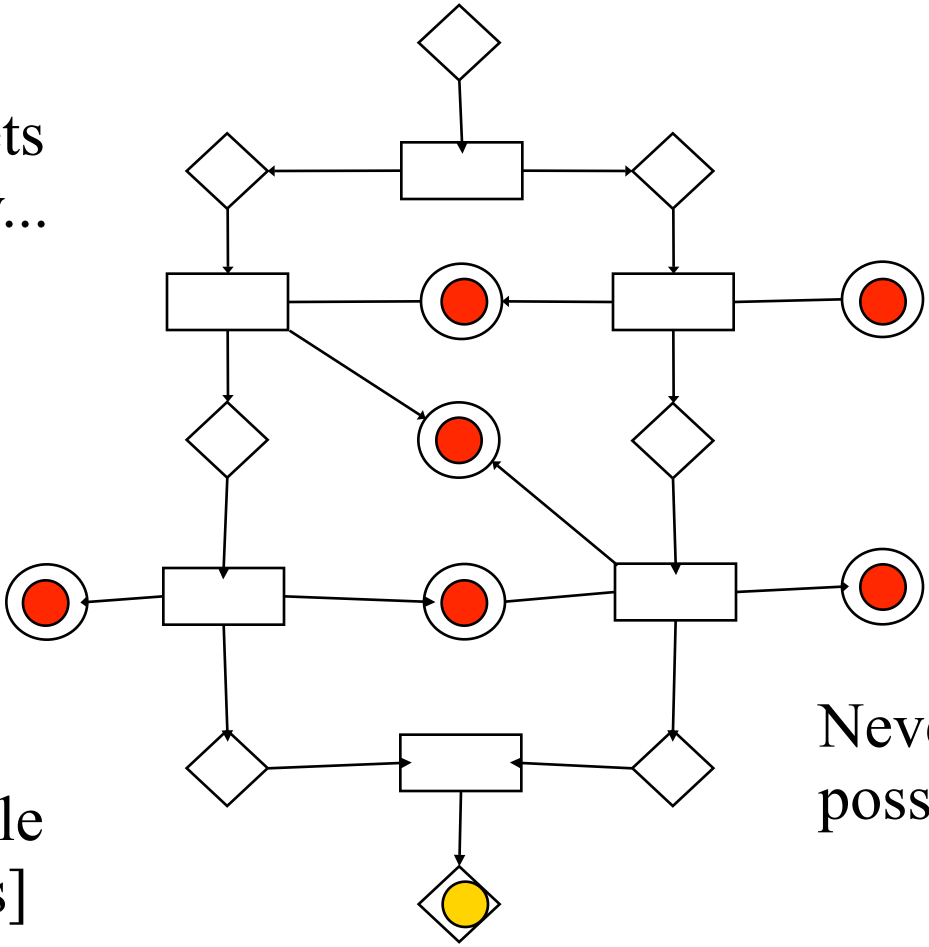


Never consumed, and
possibly overlapped!

The token game

Behave as C/E nets
w.r.t. control flow...

but data places
are persistent!



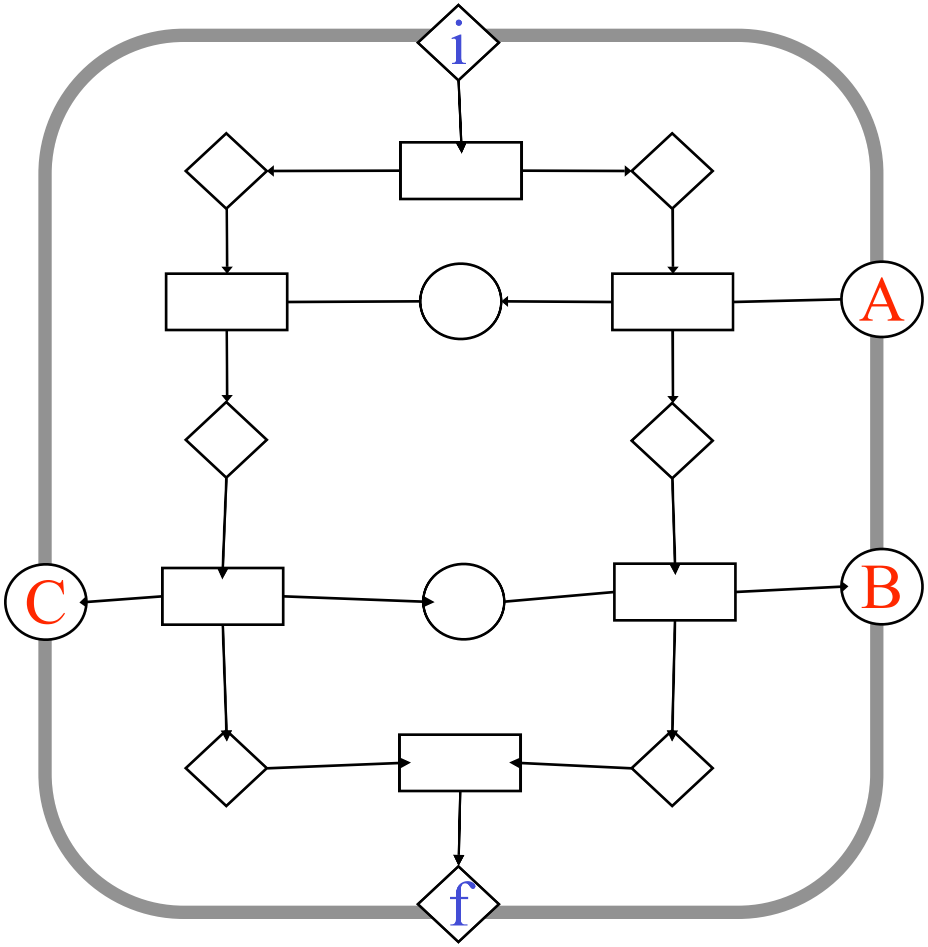
Never consumed, and
possibly overlapped!

[Usually reachable
- as e.g. WF nets]

Plan of the talk

- From Web Services to CPR nets
- **Open CPR nets and Contexts**
- Saturated Semantics for OCPR nets
- Conclusions (and example)

Open CPR nets



Open CPR net

=

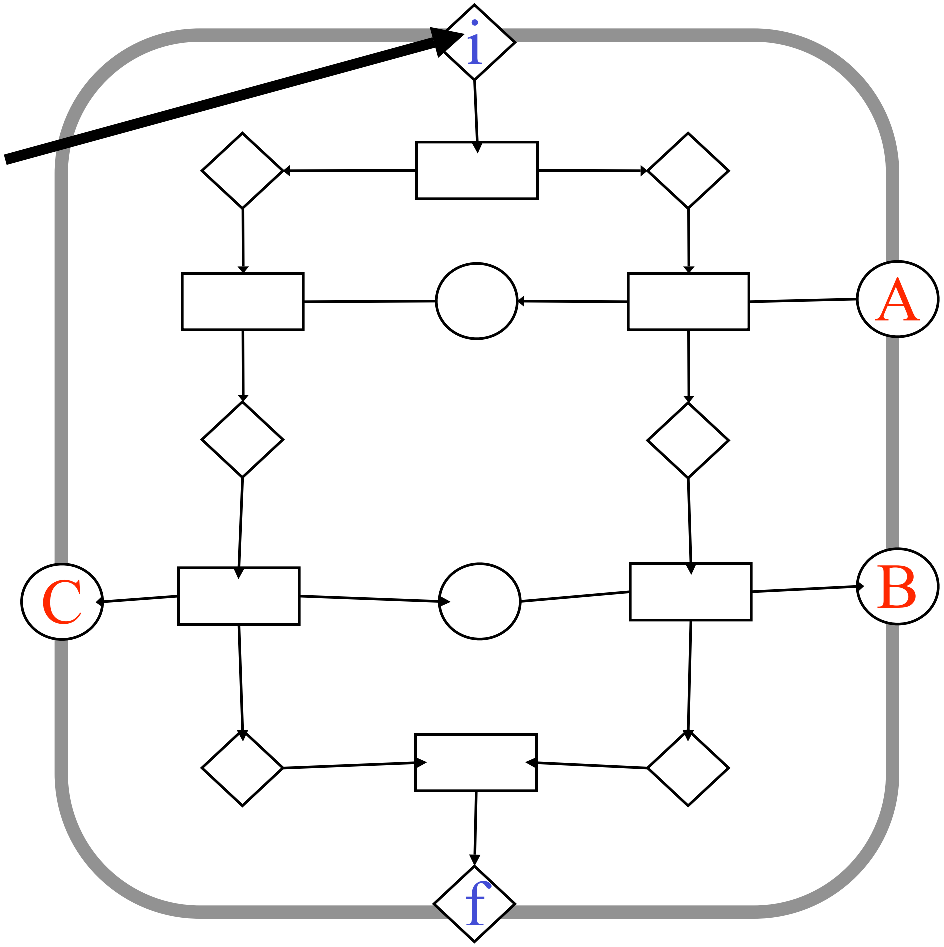
CPR net

+

Outer Interface

Open CPR nets

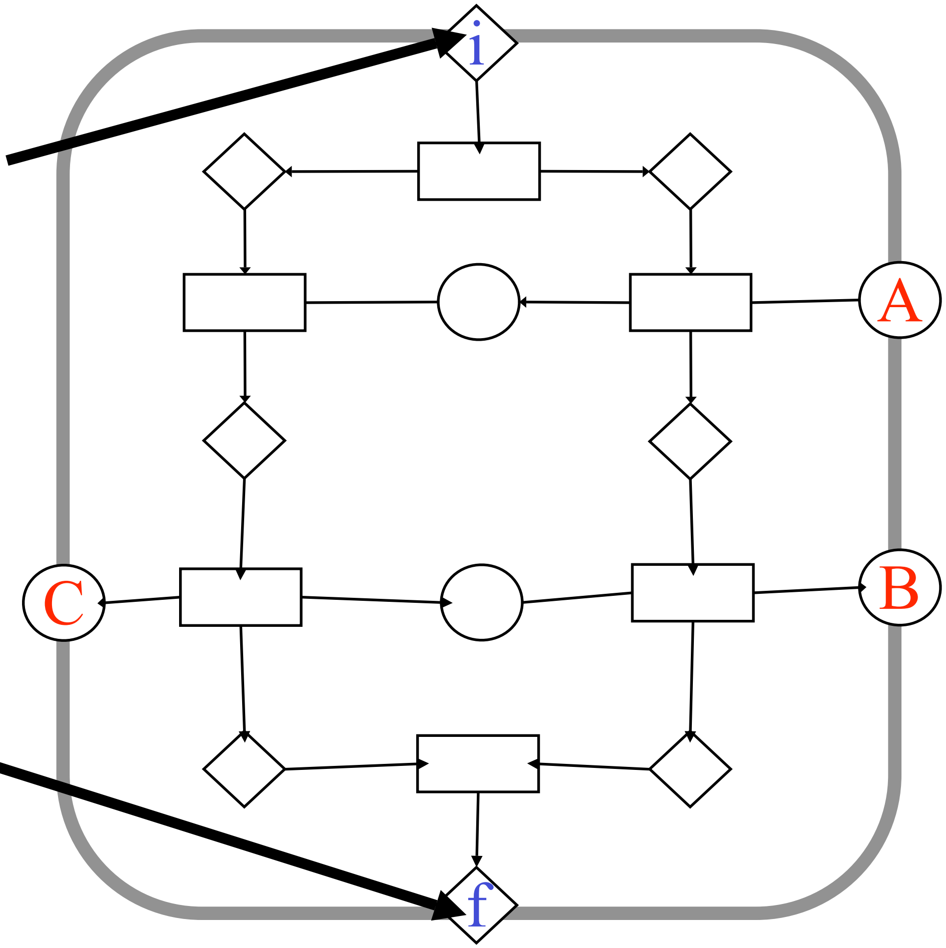
Initial Place (no incoming transitions)



Open CPR net
=
CPR net
+
Outer Interface

Open CPR nets

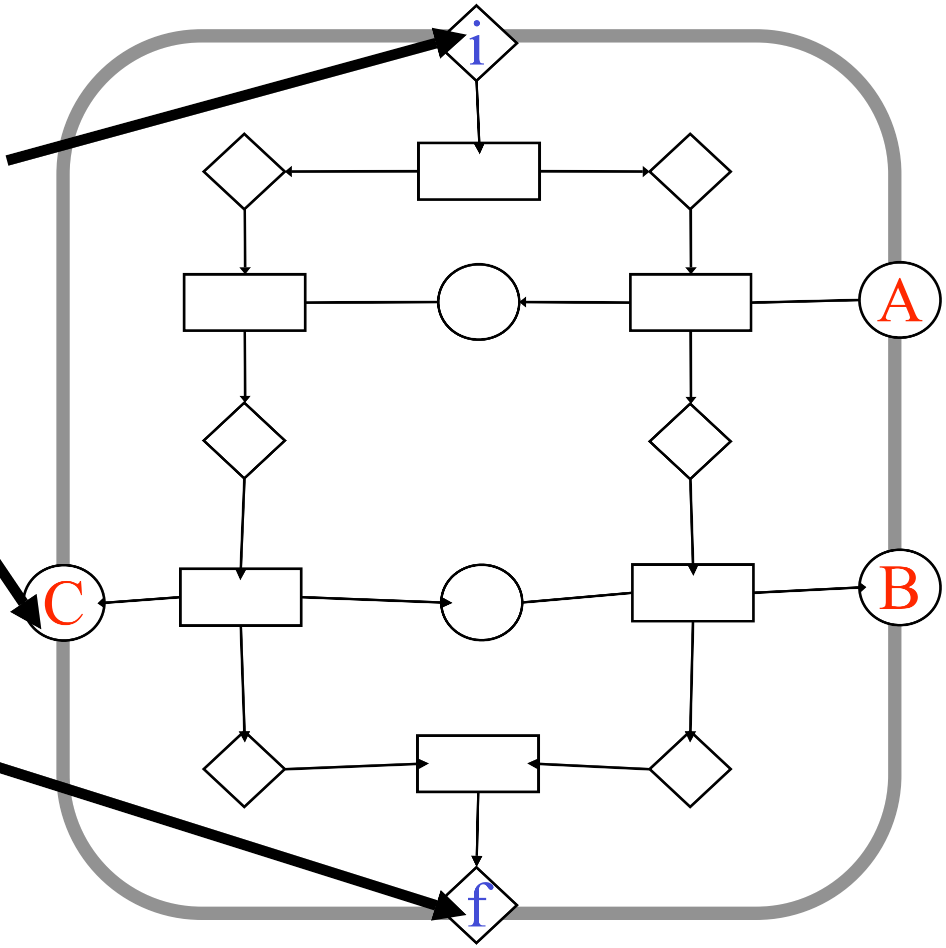
Initial Place
(no incoming
transitions)



Final Place
(no outgoing
transitions)

Open CPR net
=
CPR net
+
Outer
Interface

Open CPR nets



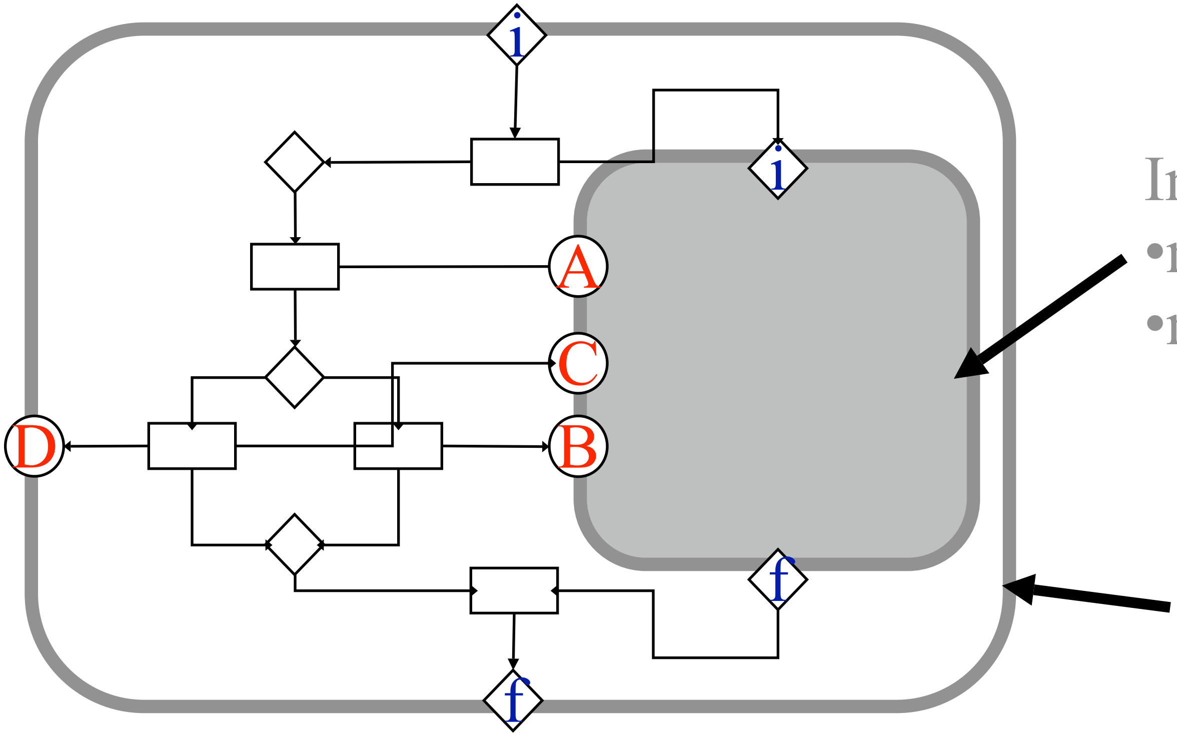
Initial Place (no incoming transitions)

Open Data Places

Final Place (no outgoing transitions)

Open CPR net = CPR net + Outer Interface

CPR-Contexts

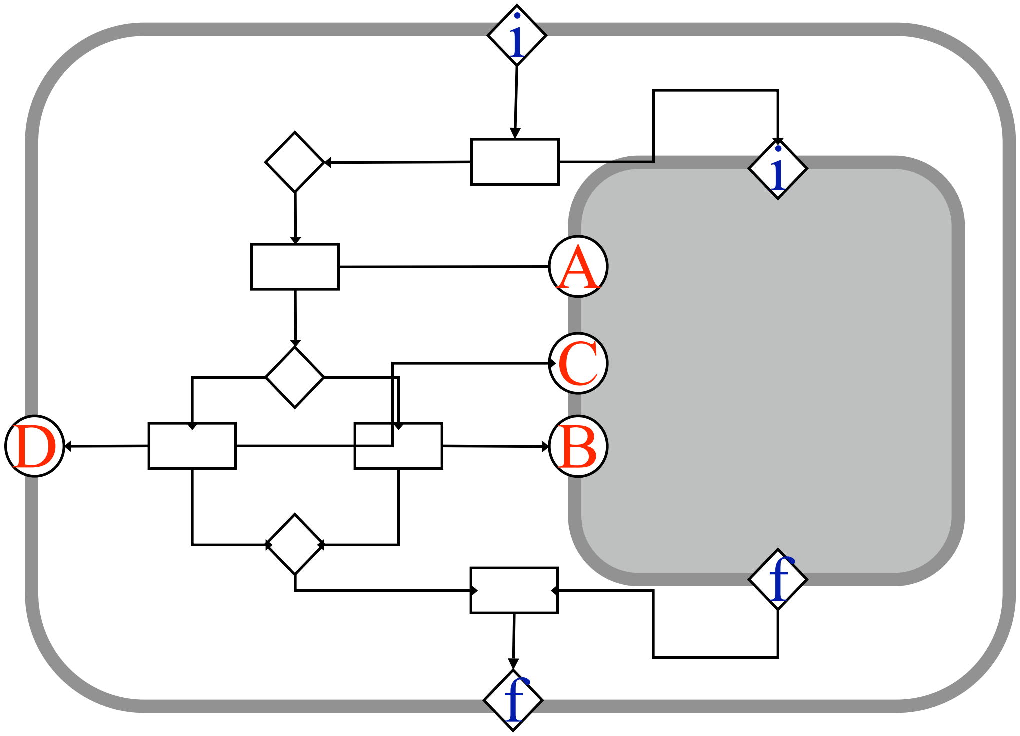


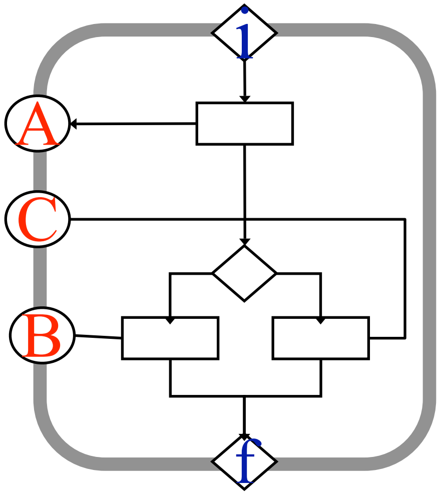
Inner Interface

- no outgoing arc into i
- no incoming arc into f

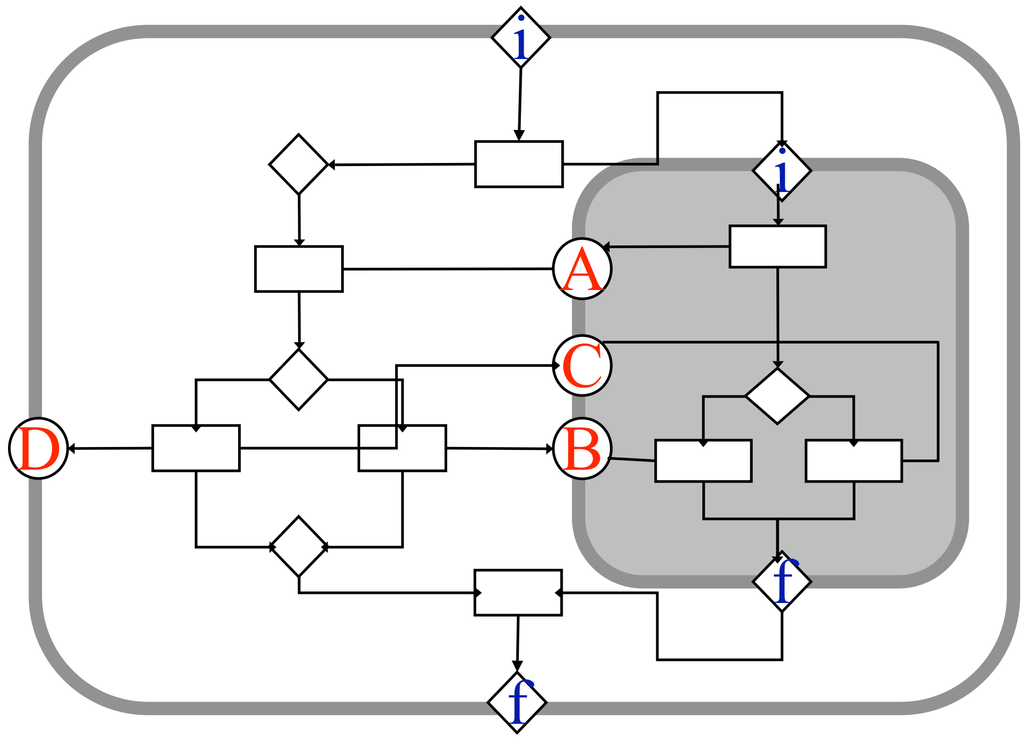
Outer Interface

Inserting nets into contexts

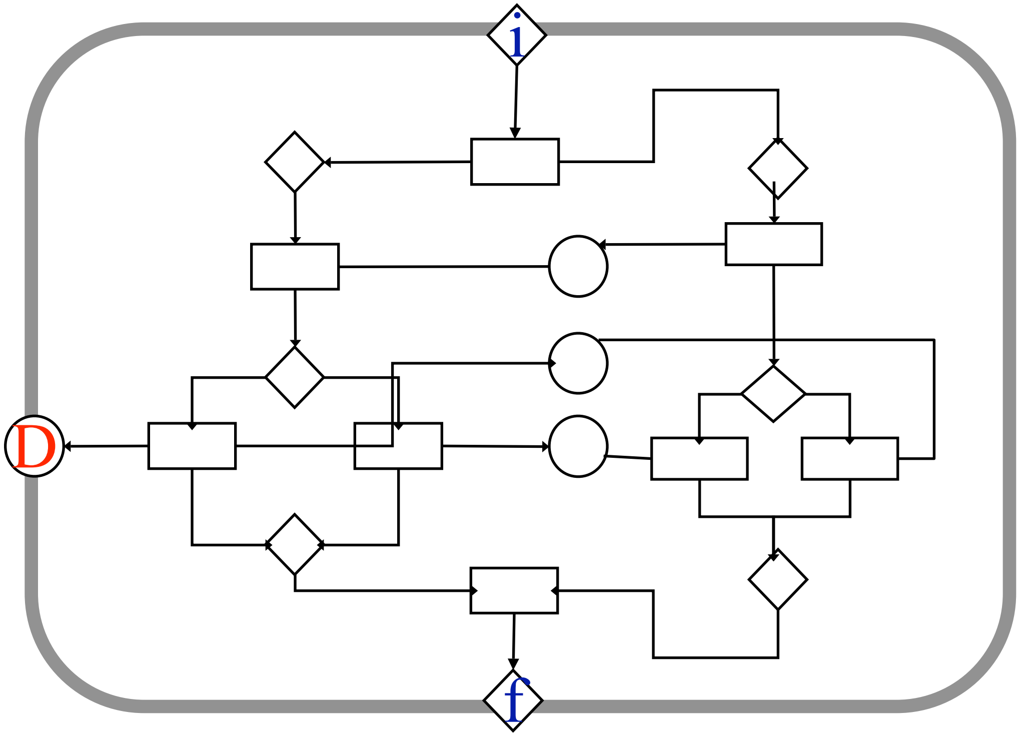




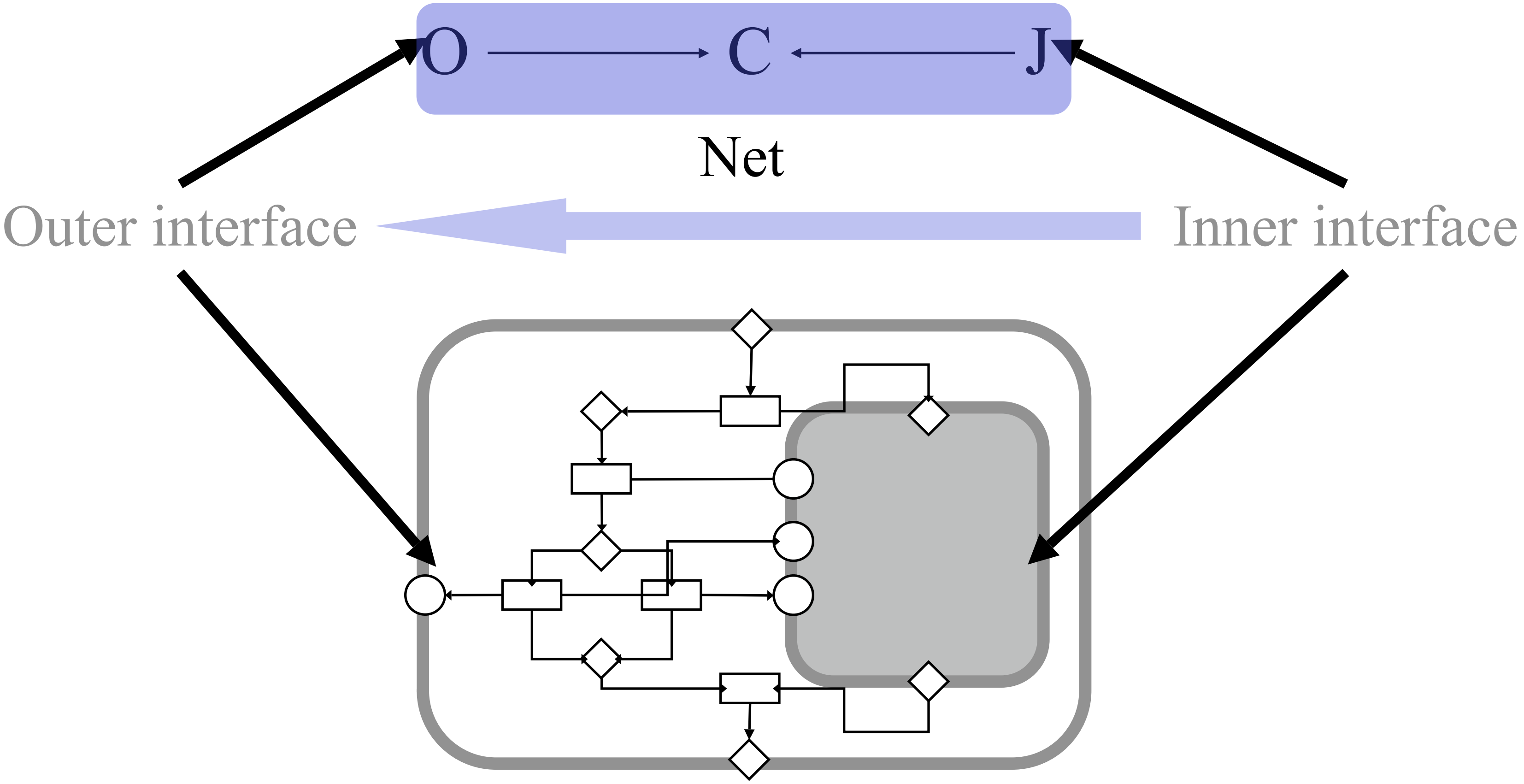
Inserting nets into contexts



Inserting nets into contexts



Contexts as Category of Cospans



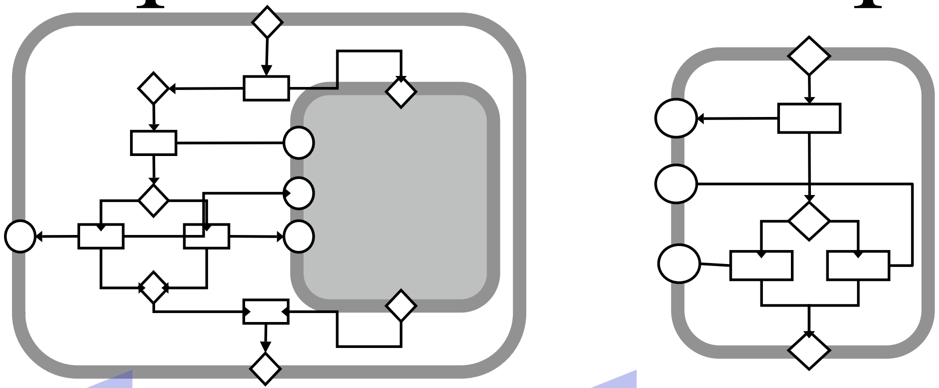
$$\begin{tikzcd} O \arrow[r] & C & J \arrow[l] \end{tikzcd}$$

Net

Outer interface

Inner interface

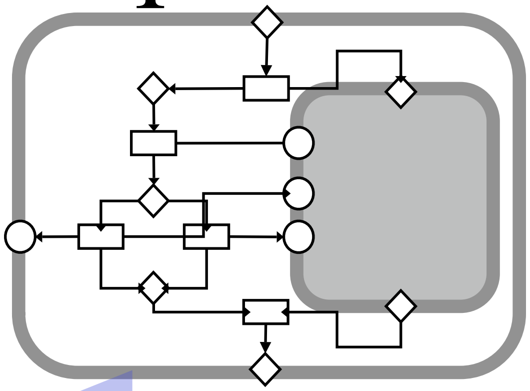
Composition of Cospans

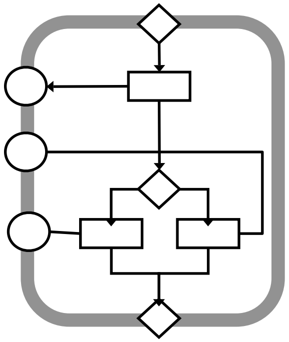


$$\begin{tikzcd}
O \arrow[r] & C & J \arrow[l] \arrow[r] & N & \emptyset \arrow[l]
\end{tikzcd}$$

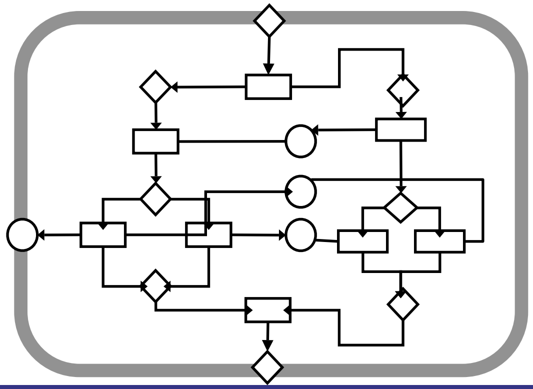
Inner interface is the empty set

Composition of Cospans





$$\begin{tikzcd}
O \arrow[r] & C \arrow[dr] & J \arrow[l] \arrow[r] \arrow[d, phantom, "PO"] & N \arrow[dl] & \emptyset \arrow[l] \\
& & C +_J N & &
\end{tikzcd}$$



Inner interface is the empty set

Plan of the talk

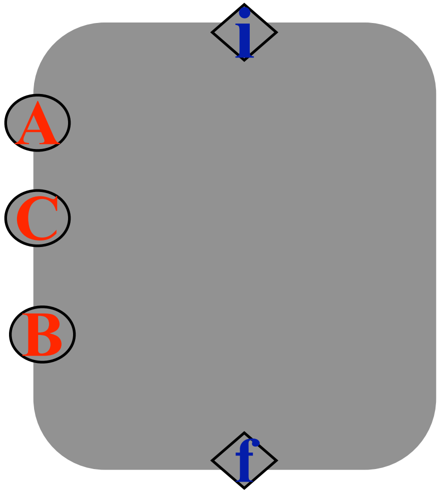
- From Web Services to CPR nets
- Open CPR nets and Contexts
- **Saturated Semantics for CPR nets**
- Conclusions (and example)

Motivations

- Incremental development of services
- Matching of (composition of) services
- Publishing of services
- Replacing of services

➡ WEAK & COMPOSITIONAL

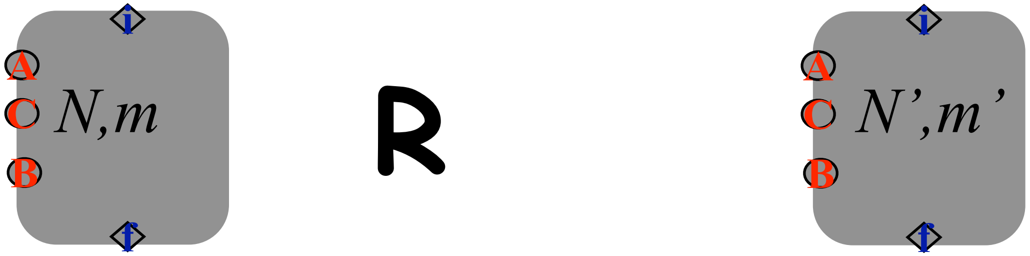
Basic Observations



Given an Open CPR net *N*,
and a marking *m*,

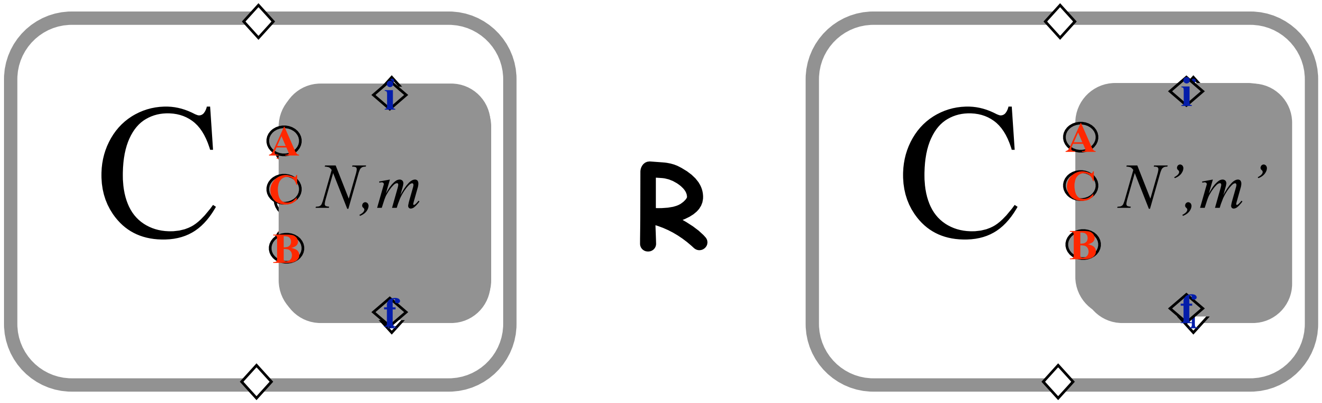
Obs(N,m) =
tokens in the open places

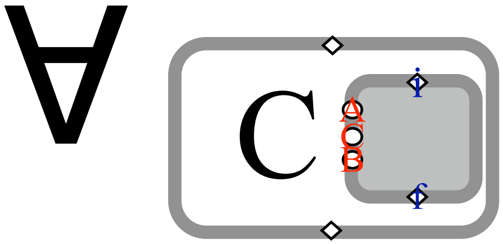
Saturated Bisimulation



$$Obs(N,m) = Obs(N',m')$$

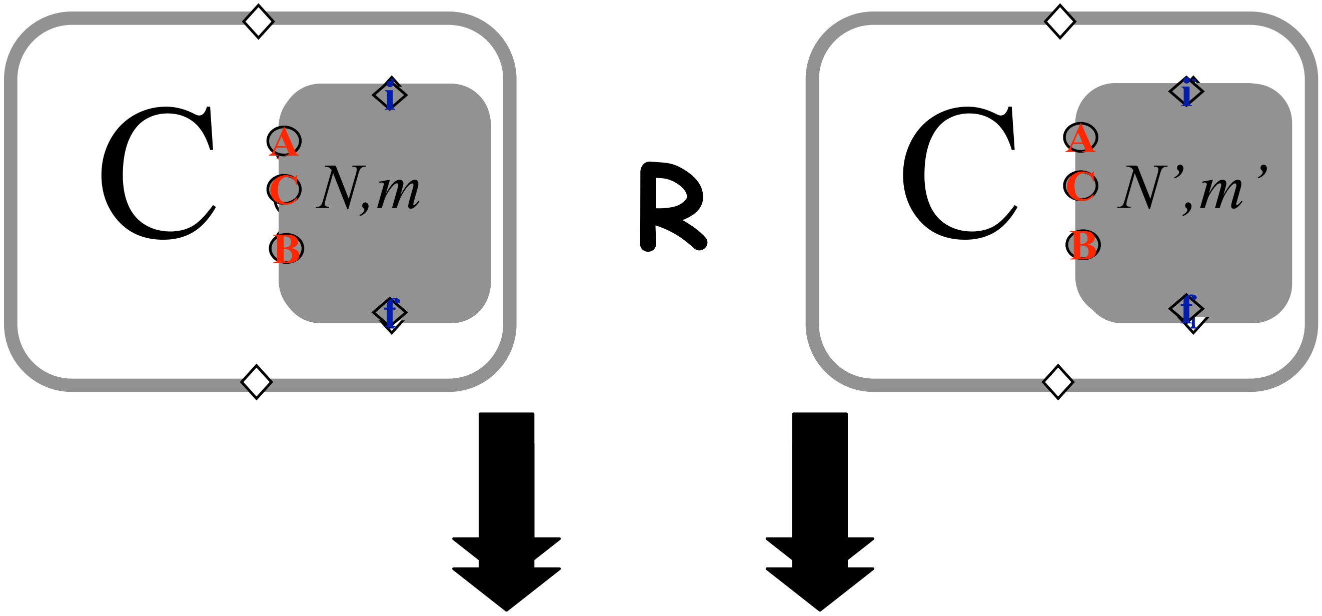
Saturated Bisimulation

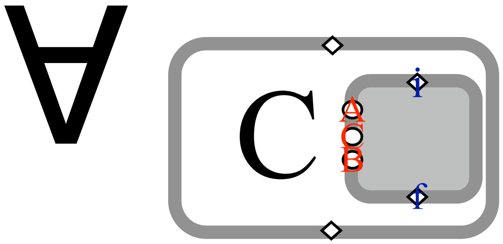




$$Obs(N,m) = Obs(N',m')$$

Saturated Bisimulation





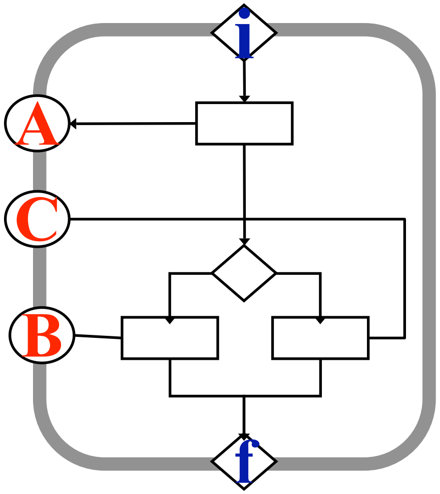
C[N],o R *C[N']*,*o'*

$$Obs(N,m) = Obs(N',m')$$

Interactive Semantics

From reduction semantics (firing) we derive a LTS

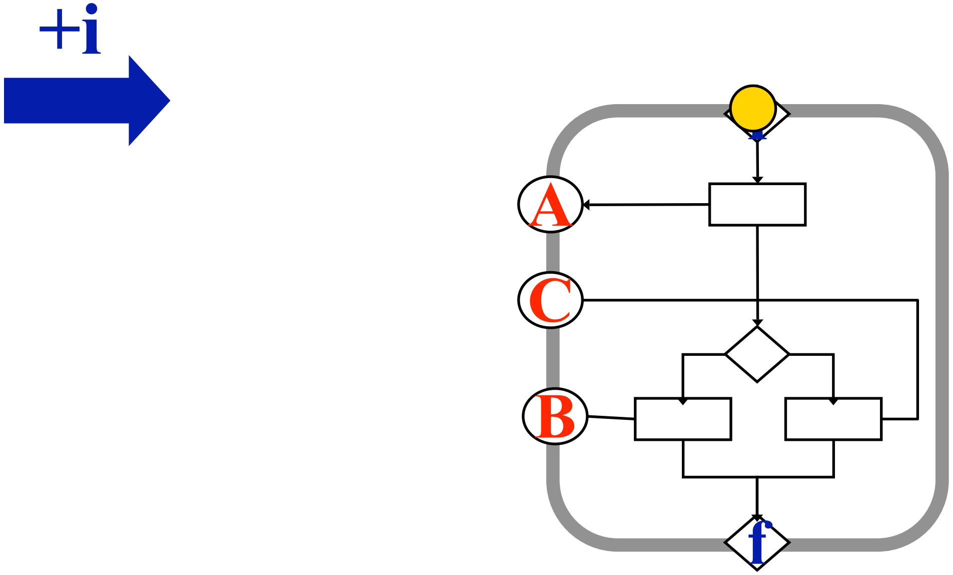
Labels model
the interactions
of the net
with the environment



Tokens can be
added and/or
removed
from open places

Interactive Semantics

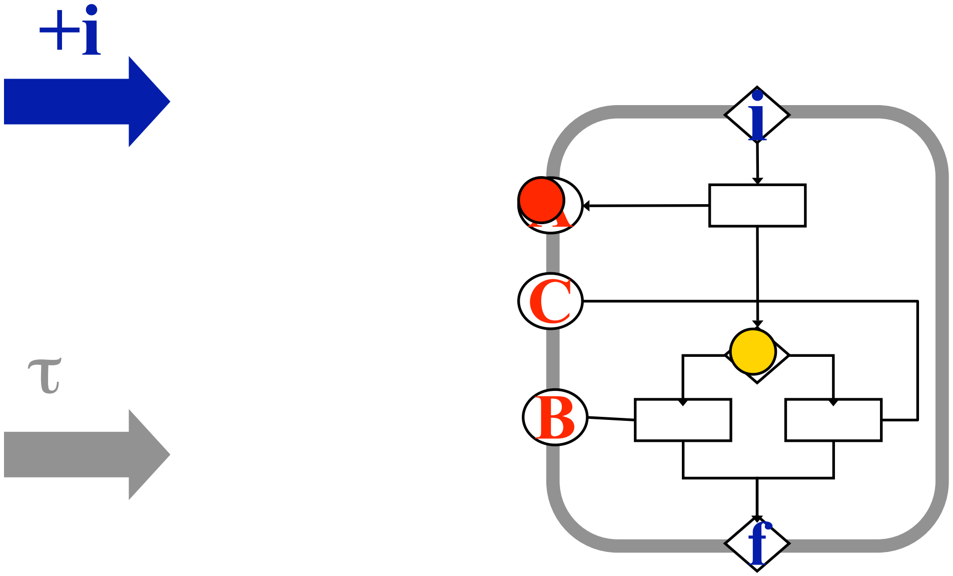
From reduction semantics (firing) we derive a LTS



Tokens can be
added and/or
removed
from open places

Interactive Semantics

From reduction semantics (firing) we derive a LTS



Tokens can be
added and/or
removed
from open places

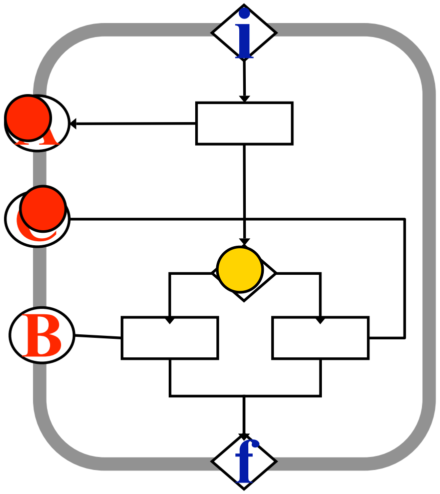
Interactive Semantics

From reduction semantics (firing) we derive a LTS

+i

τ

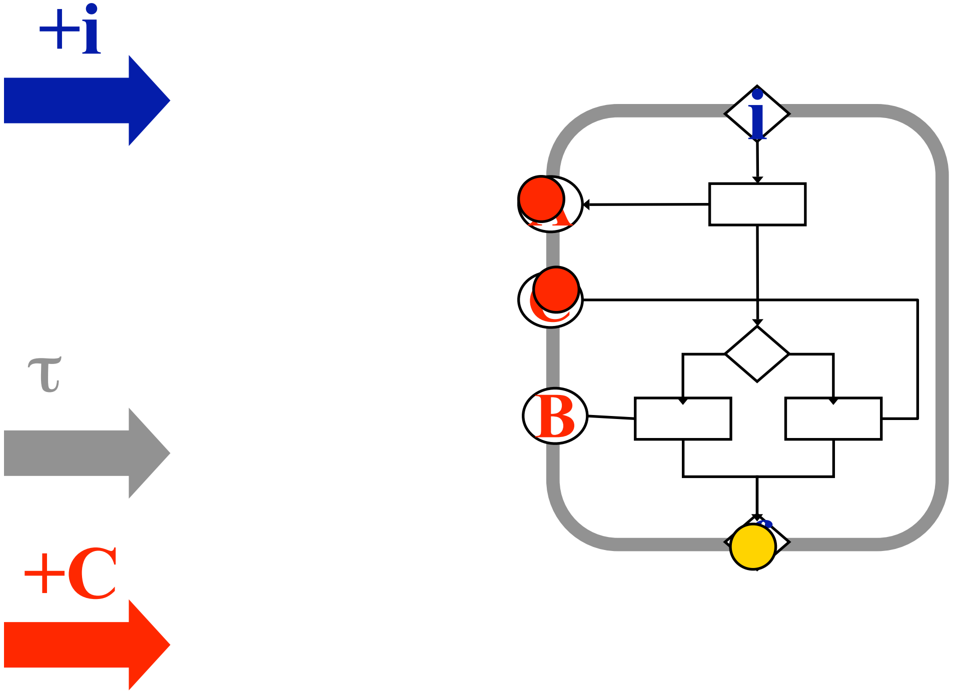
+C



Tokens can be
added and/or
removed
from open places

Interactive Semantics

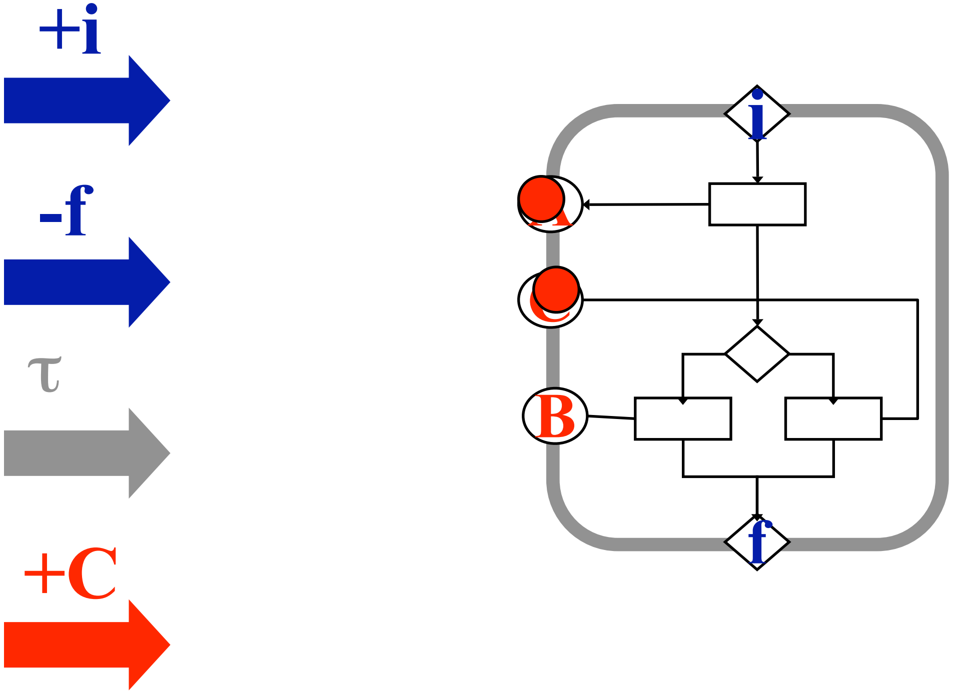
From reduction semantics (firing) we derive a LTS



Tokens can be
added and/or
removed
from open places

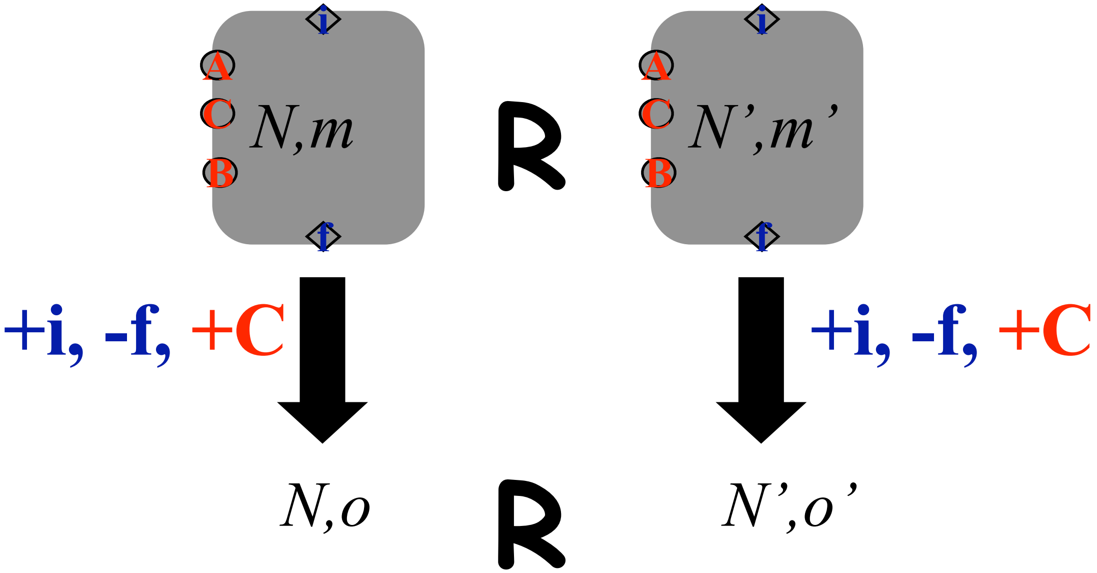
Interactive Semantics

From reduction semantics (firing) we derive a LTS



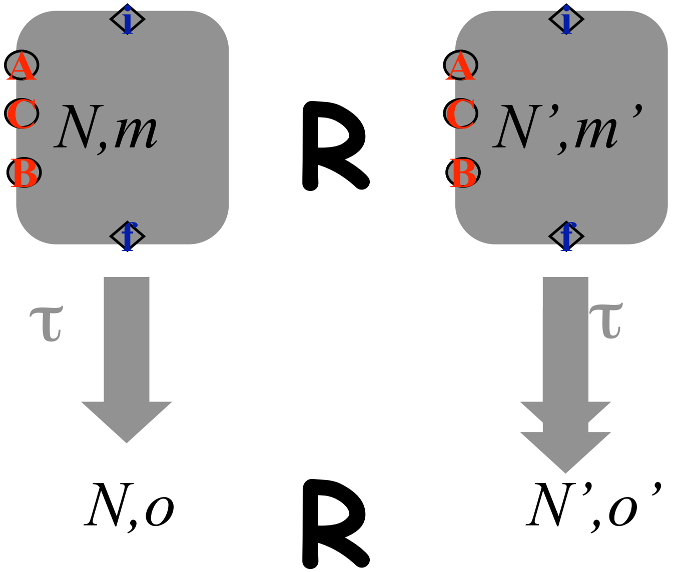
Tokens can be
added and/or
removed
from open places

Weak Bisimulation



$$Obs(N,m) = Obs(N',m')$$

Weak Bisimulation



$$Obs(N,m) = Obs(N',m')$$

Conclusions

Weak Bisimilarity

coincides with

Saturated Bisimilarity

It is

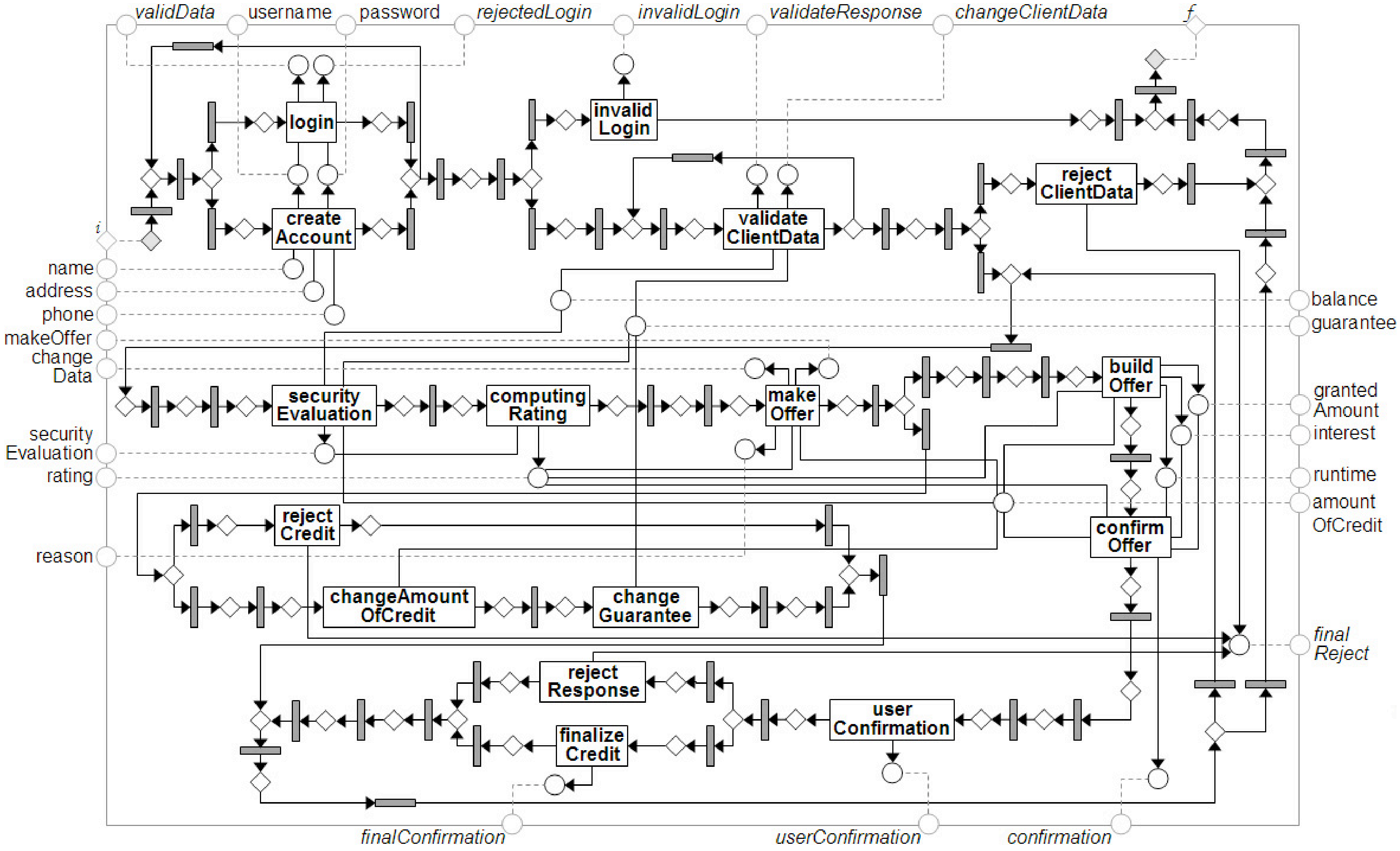
1. Weak (wrt internal transitions)
2. Compositional (congruence)
3. Computable (data persistency)

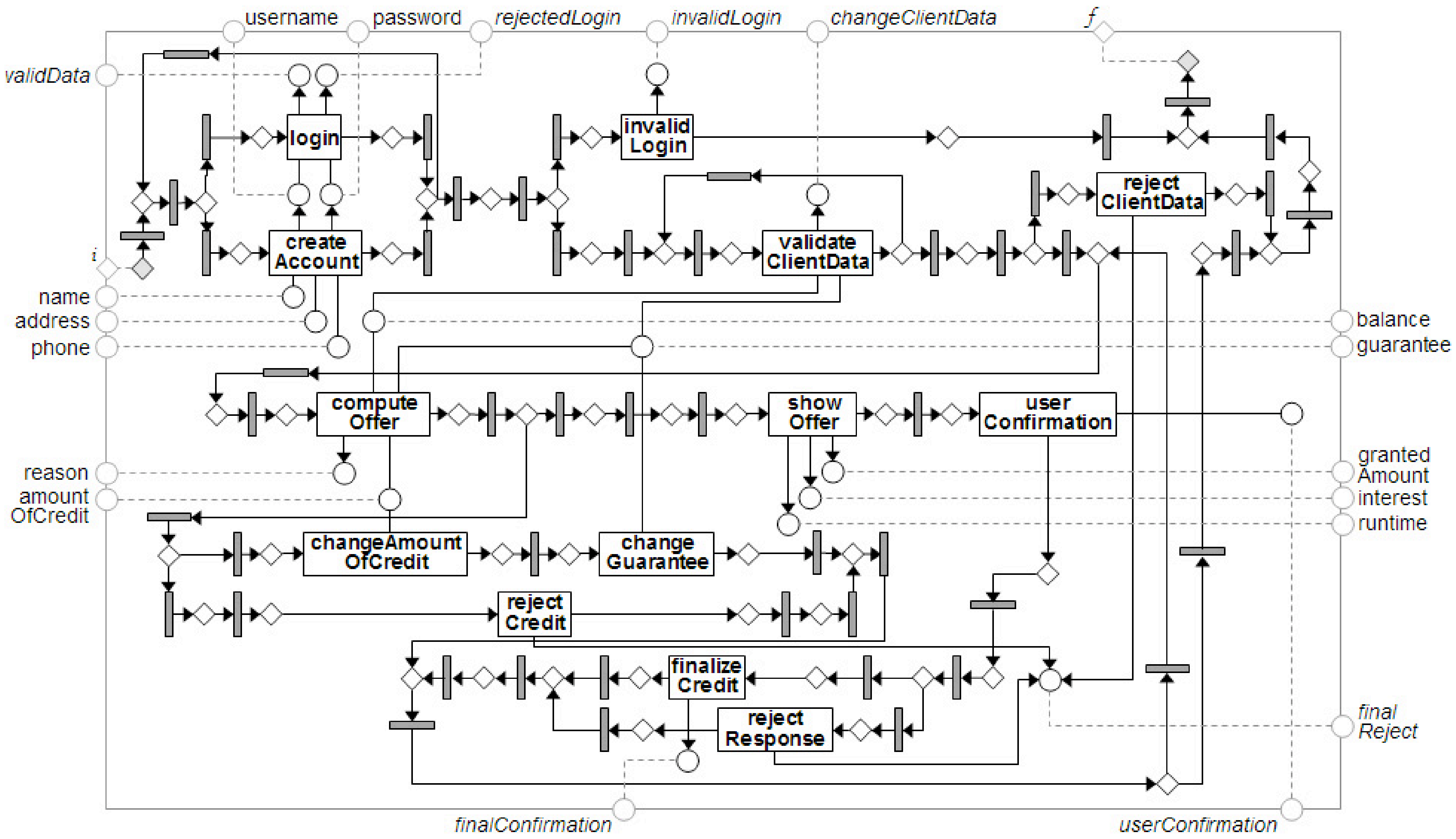
Plan of the talk

- From Web Services to CPR nets
- Open CPR nets and Contexts
- Saturated Semantics for CPR nets
- **Conclusions (and example)**

Ongoing work

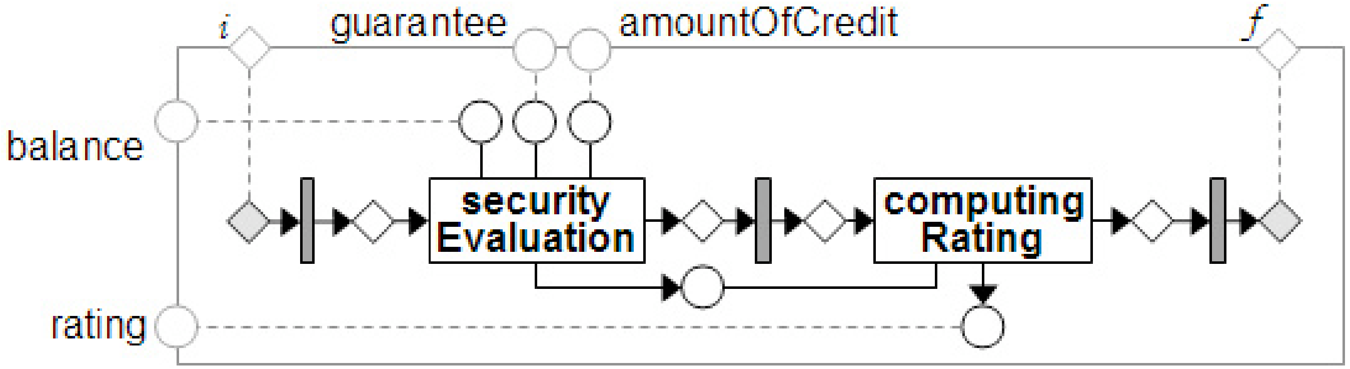
- So far...
  - A tool for mapping an OWS-L service model specification into an OCPR net
  - A straightforward adaptation of the algorithm for the verification of weak bisimularity
- Forthcoming...
  - A better adaptation of the verification algorithm
  - A generalization to other classes of nets

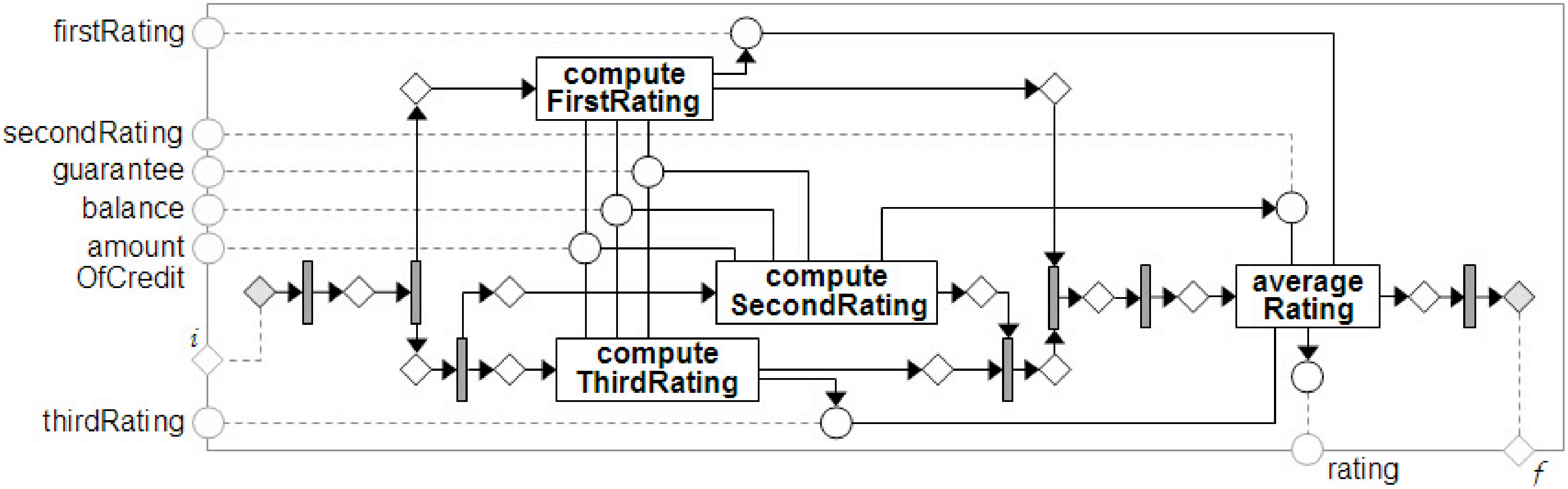

validData
username
password
rejectedLogin
invalidLogin
validateResponse
changeClientData
f
i
login
invalid Login
create Account
validate ClientData
reject ClientData
name
address
phone
makeOffer
change Data
security Evaluation
computing Rating
make Offer
build Offer
balance
guarantee
granted Amount
interest
runtime
amount OfCredit
security Evaluation
rating
reason
reject Credit
changeAmount OfCredit
change Guarantee
confirm Offer
final Reject
reject Response
finalize Credit
user Confirmation
finalConfirmation
userConfirmation
confirmation


username
password
rejectedLogin
invalidLogin
changeClientData
f
validData
login
invalid Login
i
create Account
validate ClientData
reject ClientData
name
address
phone
balance
guarantee
compute Offer
show Offer
user Confirmation
reason
amount OfCredit
granted Amount
interest
runtime
changeAmount OfCredit
change Guarantee
reject Credit
finalize Credit
reject Response
final Reject
finalConfirmation
userConfirmation

equivalent nets...

- the two net encodings (the service and its public specification) are equivalent, after removing private names from open places


i
guarantee
amountOfCredit
f
balance
security Evaluation
computing Rating
rating


firstRating
compute FirstRating
secondRating
guarantee
balance
amount OfCredit
compute SecondRating
average Rating
i
compute ThirdRating
thirdRating
rating
f

equivalent nets...

- the two net encodings (the service and its public specification) are equivalent, after removing private names from open places
- the two sub-nets are equivalent, after removing the additional names [hence, the new one can be safely plugged in...]