pre-kappa expander for κ language

Héctor Urbina

July 19, 2011

What is κ

κ is a formal language for defining agents as sets of sites.

Sites hold an internal state as well as a binding state.

κ also enables the expression of rules of interaction between agents.

These rules are executable, inducing a stochastic dynamics on a mixture of agents.

A κ model is a collection of rules (with rate constants) and an initial mixture of agents on which such rules begin to act.

Krivine et. al. Programs as models: Kappa language basics. Unpublised work.

What is κ

κ is a formal language for defining agents as sets of sites.

Sites hold an internal state as well as a binding state.

κ also enables the expression of rules of interaction between agents.

These rules are executable, inducing a stochastic dynamics on a mixture of agents.

A κ model is a collection of rules (with rate constants) and an initial mixture of agents on which such rules begin to act.

Krivine et. al. Programs as models: Kappa language basics. Unpublised work.

What is κ

κ is a formal language for defining agents as sets of sites.

Sites hold an internal state as well as a binding state.

κ also enables the expression of rules of interaction between agents.

These rules are executable, inducing a stochastic dynamics on a mixture of agents.

A κ model is a collection of rules (with rate constants) and an initial mixture of agents on which such rules begin to act.

Krivine et. al. Programs as models: Kappa language basics. Unpublised work.

What is κ

κ is a formal language for defining agents as sets of sites.

Sites hold an internal state as well as a binding state.

κ also enables the expression of rules of interaction between agents.

These rules are executable, inducing a stochastic dynamics on a mixture of agents.

A κ model is a collection of rules (with rate constants) and an initial mixture of agents on which such rules begin to act.

Krivine et. al. Programs as models: Kappa language basics. Unpublised work.

What is κ

κ is a formal language for defining agents as sets of sites.

Sites hold an internal state as well as a binding state.

κ also enables the expression of rules of interaction between agents.

These rules are executable, inducing a stochastic dynamics on a mixture of agents.

A κ model is a collection of rules (with rate constants) and an initial mixture of agents on which such rules begin to act.

Krivine et. al. Programs as models: Kappa language basics. Unpublised work.

κ Syntax short introduction

Rule in English:

"Unphosphorilated Site1 of A binds to Site1 of B."

κ Rule:

A(Site1~u),B(Site1) $\rightarrow$ A(Site1~u!1),B(Site1!1)

- Agent Names : an identifier.
- Agent Sites : an identifier.
- Internal States : ~$\langle$value$\rangle$.
- Binding States : !$\langle$n$\rangle$, !_ or ?.

κ Syntax short introduction

Rule in English:

"Unphosphorilated Site1 of A binds to Site1 of B."

κ Rule:

A(Site1~u),B(Site1) $\rightarrow$ A(Site1~u!1),B(Site1!1)

- Agent Names : an identifier.
- Agent Sites : an identifier.
- Internal States : ~$\langle$value$\rangle$.
- Binding States : !$\langle$n$\rangle$, !_ or ?.

κ Syntax short introduction

Rule in English:

"Unphosphorilated Site1 of A binds to Site1 of B."

κ Rule:

A(Site1~u),B(Site1) $\rightarrow$ A(Site1~u!1),B(Site1!1)

- Agent Names : an identifier.
- Agent Sites : an identifier.
- Internal States : ~$\langle$value$\rangle$.
- Binding States : !$\langle$n$\rangle$, !_ or ?.

Kappa file structure


```
#### Signatures
%agent:  A(x,c) # Declaration of agent A
%agent:  B(x) # Declaration of B
%agent:  C(x1~u~p,x2~u~p) # Declaration of C with 2 modifiable sites
#### Rules
‘a.b’ A(x),B(x) -> A(x!1),B(x!1) @ ‘on_rate’ #A binds B
‘a..b’ A(x!1),B(x!1) -> A(x),B(x) @ ‘off_rate’ #AB dissociation
’ab.c’ A(x!_,c),C(x1~u) ->A(x!_,c!2),C(x1~u!2) @ ‘on_rate’ #AB binds C
’mod x1’ C(x1~u!1),A(c!1) ->C(x1~p),A(c) @ ‘mod_rate’ #AB modifies x1
’a.c’ A(x,c),C(x1~p,x2~u) -> A(x,c!1),C(x1~p,x2~u!1) @ ‘on_rate’ #A binds C on x2
’mod x2’ A(x,c!1),C(x1~p,x2~u!1) -> A(x,c),C(x1~p,x2~p) @ ‘mod_rate’ #A modifies x2
#### Variables
%var:  ‘on_rate’ 1.0E-4 # per molecule per second
%var:  ‘off_rate’ 0.1 # per second
%var:  ‘mod_rate’ 1 # per second
%obs:  ‘AB’ A(x!x.B)
%obs:  ‘Cuu’ C(x1~u,x2~u)
%obs:  ‘Cpu’ C(x1~p,x2~u)
%obs:  ‘Cpp’ C(x1~p,x2~p)
#### Initial conditions
%init:  1000 A,B
%init:  10000 C
```

KaSim reference manual v1.06

DLab's current work

DLab members study complex dynamical systems.

Currently, Cesar Ravello is modeling muscle contration and Felipe Nuñez is simulating massive responses to zombie attacks on human populations, whereas Ricardo Honorato is adapting Model Checking techniques to be used with systems expressed in κ language.

Without intervening the κ language, we have reached some interesting levels of abstraction!

DLab's current work

DLab members study complex dynamical systems.

Currently, Cesar Ravello is modeling muscle contration and Felipe Nuñez is simulating massive responses to zombie attacks on human populations, whereas Ricardo Honorato is adapting Model Checking techniques to be used with systems expressed in κ language.

Without intervening the κ language, we have reached some interesting levels of abstraction!

DLab's current work

DLab members study complex dynamical systems.

Currently, Cesar Ravello is modeling muscle contration and Felipe Nuñez is simulating massive responses to zombie attacks on human populations, whereas Ricardo Honorato is adapting Model Checking techniques to be used with systems expressed in κ language.

Without intervening the κ language, we have reached some interesting levels of abstraction!

DLab's current work

- Space-related simulations.
  - Compartmentalization.
  - Diffusion events.
- Timming control.
  - Polymer-driven rules to manipulate latency.

DLab's current work

- Space-related simulations.
  - Compartmentalization.
  - Diffusion events.

- Timming control.
  - Polymer-driven rules to manipulate latency.

DLab's current work

- Space-related simulations.
  - Compartmentalization.
  - Diffusion events.
- Timming control.
  - Polymer-driven rules to manipulate latency.

Compartmentalization

#Signatures
%agent: A(x,c,loc~i~j~k)
%agent: B(x,loc~i~j~k)

#Rules

#A binds B

A(x,loc~i),B(x,loc~i) $\rightarrow$ A(x!1,loc~i),B(x!1,loc~i) @ 'on_rate'

A(x,loc~j),B(x,loc~j) $\rightarrow$ A(x!1,loc~j),B(x!1,loc~j) @ 'on_rate'

A(x,loc~k),B(x,loc~k) $\rightarrow$ A(x!1,loc~k),B(x!1,loc~k) @ 'on_rate'

Compartmentalization

#Signatures
%agent: A(x,c,loc~i~j~k)
%agent: B(x,loc~i~j~k)

#Rules

#A binds B

A(x,loc~i),B(x,loc~i) → A(x!1,loc~i),B(x!1,loc~i) @ 'on_rate'

A(x,loc~j),B(x,loc~j) → A(x!1,loc~j),B(x!1,loc~j) @ 'on_rate'

A(x,loc~k),B(x,loc~k) → A(x!1,loc~k),B(x!1,loc~k) @ 'on_rate'

Compartmentalization

#Locations i, j and k have different volumen/area!
#Signatures
%agent: A(x,c,loc~i~j~k)
%agent: B(x,loc~i~j~k)

#Rules

#A binds B

A(x,loc~i),B(x,loc~i) → A(x!1,loc~i),B(x!1,loc~i) @ 'on_rate_loc(i)'

A(x,loc~j),B(x,loc~j) → A(x!1,loc~j),B(X!1,loc~j) @ 'on_rate_loc(j)'

A(x,loc~k),B(x,loc~k) → A(x!1,loc~k),B(X!1,loc~k) @ 'on_rate_loc(k)'

#AB dissociation

A(x!1,loc~i),B(x!1,loc~i) → A(x,loc~i),B(x,loc~i) @ 'off_rate'

A(x!1,loc~j),B(x!1,loc~j) → A(x,loc~j),B(x,loc~j) @ 'off_rate'

A(x!1,loc~k),B(x!1,loc~k) → A(x,loc~k),B(x,loc~k) @ 'off_rate'

Compartmentalization

#Locations i, j and k have different volumen/area!
#Signatures
%agent: A(x,c,loc~i~j~k)
%agent: B(x,loc~i~j~k)

#Rules

#A binds B

A(x,loc~i),B(x,loc~i) → A(x!1,loc~i),B(x!1,loc~i) @ 'on_rate_loc(i)'

A(x,loc~j),B(x,loc~j) → A(x!1,loc~j),B(X!1,loc~j) @ 'on_rate_loc(j)'

A(x,loc~k),B(x,loc~k) → A(x!1,loc~k),B(X!1,loc~k) @ 'on_rate_loc(k)'

#AB dissociation

A(x!1,loc~i),B(x!1,loc~i) → A(x,loc~i),B(x,loc~i) @ 'off_rate'

A(x!1,loc~j),B(x!1,loc~j) → A(x,loc~j),B(x,loc~j) @ 'off_rate'

A(x!1,loc~k),B(x!1,loc~k) → A(x,loc~k),B(x,loc~k) @ 'off_rate'

Compartmentalization

```
#Locations i, j and k have different volumen/area!
#Signatures
%agent: A(x,c,loc~i~j~k)
%agent: B(x,loc~i~j~k)

#Rules

#A binds B

A(x,loc~i),B(x,loc~i) → A(x!1,loc~i),B(x!1,loc~i) @ 'on_rate_loc(i)'

A(x,loc~j),B(x,loc~j) → A(x!1,loc~j),B(X!1,loc~j) @ 'on_rate_loc(j)'

A(x,loc~k),B(x,loc~k) → A(x!1,loc~k),B(X!1,loc~k) @ 'on_rate_loc(k)'

#AB dissociation

A(x!1,loc~i),B(x!1,loc~i) → A(x,loc~i),B(x,loc~i) @ 'off_rate'

A(x!1,loc~j),B(x!1,loc~j) → A(x,loc~j),B(x,loc~j) @ 'off_rate'

A(x!1,loc~k),B(x!1,loc~k) → A(x,loc~k),B(x,loc~k) @ 'off_rate'
```

Diffusion events

```
#Signatures
%agent: A(x,c,loc~i~j~k)
%agent: B(x,loc~i~j~k)
%agent: T(s,org~i~j~k,dst~i~j~k)
```

Diffusion events

#Rules

#A diffusions

```
A(loc~i,x,c),T(org~i,dst~j) → A(loc~j,x,c),T(org~i,dst~j) @ 'Adiff_ij'
A(loc~i,x,c),T(org~i,dst~k) → A(loc~k,x,c),T(org~i,dst~k) @ 'Adiff_ik'
A(loc~j,x,c),T(org~j,dst~i) → A(loc~i,x,c),T(org~j,dst~i) @ 'Adiff_ji'
A(loc~j,x,c),T(org~j,dst~k) → A(loc~k,x,c),T(org~j,dst~k) @ 'Adiff_jk'
A(loc~k,x,c),T(org~k,dst~i) → A(loc~i,x,c),T(org~k,dst~i) @ 'Adiff_ki'
A(loc~k,x,c),T(org~k,dst~j) → A(loc~j,x,c),T(org~k,dst~j) @ 'Adiff_kj'
```

polymer-driven rules

#Signatures
%agent: S(x)
%agent: Z()
%agent: V(p,n)

#Rules
'Infection' Z(),S(x) $\rightarrow$ Z(),S(x!1),V(p!1,n) @ 'infection_rate'
'Polymerization' V(n) $\rightarrow$ V(n!1),V(p!1,n) @ 'polymer_rate'
'Expression' S(x!1),V(p!1,n!2),V(p!2,n!3),V(p!3,n!4), \
V(p!4,n!5),V(p!5,n!6),V(p!6,n!7),V(p!7,n!8),V(p!8,n!9), \
V(p!9,n!10),V(p!10,n) $\rightarrow$ Z() @ [inf]

polymer-driven rules

#Signatures
%agent: S(x)
%agent: Z()
%agent: V(p,n)

#Rules
'Infection' Z(),S(x) → Z(),S(x!1),V(p!1,n) @ 'infection_rate'
'Polymerization' V(n) → V(n!1),V(p!1,n) @ 'polymer_rate'
'Expression' S(x!1),V(p!1,n!2),V(p!2,n!3),V(p!3,n!4), \
V(p!4,n!5),V(p!5,n!6),V(p!6,n!7),V(p!7,n!8),V(p!8,n!9), \
V(p!9,n!10),V(p!10,n) → Z() @ [inf]

polymer-driven rules


```
#Signatures
%agent: S(x)
%agent: Z()
%agent: V(p,n)
#Rules
'Infection' Z(),S(x) → Z(),S(x!1),V(p!1,n) @ 'infection_rate'
'Polymerization' V(n) → V(n!1),V(p!1,n) @ 'polymer_rate'
'Expression' S(x!1),V(p!1,n!2),V(p!2,n!3),V(p!3,n!4), \
V(p!4,n!5),V(p!5,n!6),V(p!6,n!7),V(p!7,n!8),V(p!8,n!9), \
V(p!9,n!10),V(p!10,n) → Z() @ [inf]
```

#Signatures
%agent: S(x)
%agent: Z()
%agent: V(p,n)

#Rules
'Infection' Z(),S(x) → Z(),S(x!1),V(p!1,n) @ 'infection_rate'
'Polymerization' V(n) → V(n!1),V(p!1,n) @ 'polymer_rate'
'Expression' S(x!1),V(p!1,n!2),V(p!2,n!3),V(p!3,n!4), \
V(p!4,n!5),V(p!5,n!6),V(p!6,n!7),V(p!7,n!8),V(p!8,n!9), \
V(p!9,n!10),V(p!10,n) → Z() @ [inf]

pre-Kappa expander

A Python (V2) script that takes as input a (built in-house) pre-κ file and outputs a kappa file which can subsequently be used with KaSim.

This is done using Lexer & Parser techniques, available in Python through ply library.

It facilitates κ abstraction while reducing error-proneness.

Freely available on https://github.com/ajendrex/expander/

pre-Kappa expander

A Python (V2) script that takes as input a (built in-house) pre-κ file and outputs a kappa file which can subsequently be used with KaSim.

This is done using Lexer & Parser techniques, available in Python through ply library.

It facilitates κ abstraction while reducing error-proneness.

Freely available on https://github.com/ajendrex/expander/

pre-Kappa expander

A Python (V2) script that takes as input a (built in-house) pre-κ file and outputs a kappa file which can subsequently be used with KaSim.

This is done using Lexer & Parser techniques, available in Python through ply library.

It facilitates κ abstraction while reducing error-proneness.

Freely available on https://github.com/ajendrex/expander/

pre-Kappa expander

A Python (V2) script that takes as input a (built in-house) pre-κ file and outputs a kappa file which can subsequently be used with KaSim.

This is done using Lexer & Parser techniques, available in Python through ply library.

It facilitates κ abstraction while reducing error-proneness.

Freely available on https://github.com/ajendrex/expander/

pre-Kappa syntax: Locations

#Locations
%loc: i 100
%loc: j 1000
%loc: k 500
#Location list
%locl: all i j k
#Signatures

%expand-agent: all A(x,c)
%expand-agent: all B(x)

gives:
%agent: A(x,c,loc~i~j~k)
%agent: B(x,loc~i~j~k)

pre-Kappa syntax: Locations


```
#Locations
%loc: i 100
%loc: j 1000
%loc: k 500
#Location list
%locl: all i j k
#Signatures

%expand-agent: all A(x,c)
%expand-agent: all B(x)

gives:
%agent: A(x,c,loc~i~j~k)
%agent: B(x,loc~i~j~k)
```

pre-Kappa syntax: Locations

#Locations
%loc: i 100
%loc: j 1000
%loc: k 500
#Location list
%locl: all i j k
#Signatures

%expand-agent: all A(x,c)
%expand-agent: all B(x)

gives:
%agent: A(x,c,loc~i~j~k)
%agent: B(x,loc~i~j~k)

pre-Kappa syntax: Locations

#Locations
%loc: i 100
%loc: j 1000
%loc: k 500
#Location list
%locl: all i j k
#Signatures

%expand-agent: all A(x,c)
%expand-agent: all B(x)

gives:
%agent: A(x,c,loc~i~j~k)
%agent: B(x,loc~i~j~k)

pre-Kappa syntax: Locations


```
#Locations
%loc: i 100
%loc: j 1000
%loc: k 500
#Location list
%locl: all i j k
#Initializations (expand if densities are equal)
%expand-init: all ADensity A(x,c)
%expand-init: all BDensity B(x)
```

gives:

```
%init: ADensity * 100 A(x,c,loc~i)
%init: ADensity * 1000 A(x,c,loc~j)
%init: ADensity * 500 A(x,c,loc~k)
%init: BDensity * 100 B(x,loc~i)
%init: BDensity * 1000 B(x,loc~j)
%init: BDensity * 500 B(x,loc~k)
```

pre-Kappa syntax: Locations

#Locations
%loc: i 100
%loc: j 1000
%loc: k 500
#Location list
%locl: all i j k
#Initializations (expand if densities are equal)
%expand-init: all ADensity A(x,c)
%expand-init: all BDensity B(x)

gives:
%init: ADensity * 100 A(x,c,loc~i)
%init: ADensity * 1000 A(x,c,loc~j)
%init: ADensity * 500 A(x,c,loc~k)
%init: BDensity * 100 B(x,loc~i)
%init: BDensity * 1000 B(x,loc~j)
%init: BDensity * 500 B(x,loc~k)

pre-Kappa syntax: Locations

A bimolecular stochastic rate constant γ, expressed in $s^{-1}molecule^{-1}$, is related to its deterministic counterpart k, expressed in $s^{-1}M^{-1}$ as

$$\gamma = \frac{k}{AV}, \tag{1}$$

where A is Avogadro's number.

Krivine et. al. Programs as models: Execution. Unpublised work.

pre-Kappa syntax: Locations

#Locations
%loc: i 100
%loc: j 1000
%loc: k 500
#Location list
%locl: all i j k
#A binds B
%expand-rule: all A(x),B(x) $\rightarrow$ A(x!1),B(x!1) @ 'on_base_rate'

gives:

A(x,loc~i),B(x,loc~i) $\rightarrow$ A(x!1,loc~i),B(x!1,loc~i) @ 'on_base_rate' / 100

A(x,loc~j),B(x,loc~j) $\rightarrow$ A(x!1,loc~j),B(x!1,loc~j) @ 'on_base_rate' / 1000

A(x,loc~k),B(x,loc~k) $\rightarrow$ A(x!1,loc~k),B(x!1,loc~k) @ 'on_base_rate' / 500

pre-Kappa syntax: Locations

#Locations
%loc: i 100
%loc: j 1000
%loc: k 500
#Location list
%locl: all i j k
#A binds B
%expand-rule: all A(x),B(x) → A(x!1),B(x!1) @ 'on_base_rate'

gives:

A(x,loc~i),B(x,loc~i) → A(x!1,loc~i),B(x!1,loc~i) @ 'on_base_rate' / 100

A(x,loc~j),B(x,loc~j) → A(x!1,loc~j),B(x!1,loc~j) @ 'on_base_rate' / 1000

A(x,loc~k),B(x,loc~k) → A(x!1,loc~k),B(x!1,loc~k) @ 'on_base_rate' / 500

pre-Kappa syntax: Locations

#Locations
%loc: i 100
%loc: j 1000
%loc: k 500
#Location list
%locl: all i j k
#A binds B
%expand-rule: all A(x),B(x) → A(x!1),B(x!1) @ 'on_base_rate'

gives:

A(x,loc~i),B(x,loc~i) → A(x!1,loc~i),B(x!1,loc~i) @ 'on_base_rate' / 100

A(x,loc~j),B(x,loc~j) → A(x!1,loc~j),B(x!1,loc~j) @ 'on_base_rate' / 1000

A(x,loc~k),B(x,loc~k) → A(x!1,loc~k),B(x!1,loc~k) @ 'on_base_rate' / 500

pre-Kappa syntax: Location Matrices

...

#Location matrices

%locm:

TM	i	j	k
i	0	0.5	1.5
j	2.0	0	1.8
k	1.0	1.1	0

#A diffusions

%expand-rule: TM A(x,c),T() → A(%,x,c),T() @ 1

gives:

A(loc~i,x,c),T(org~i,dst~j) → A(loc~j,x,c),T(org~i,dst~j) @ (1 * 0.5) / 100

A(loc~i,x,c),T(org~i,dst~k) → A(loc~k,x,c),T(org~i,dst~k) @ (1 * 1.5) / 100

A(loc~j,x,c),T(org~j,dst~i) → A(loc~i,x,c),T(org~j,dst~i) @ (1 * 2.0) / 1000

A(loc~j,x,c),T(org~j,dst~k) → A(loc~k,x,c),T(org~j,dst~k) @ (1 * 1.8) / 1000

A(loc~k,x,c),T(org~k,dst~i) → A(loc~i,x,c),T(org~k,dst~i) @ (1 * 1.0) / 500

A(loc~k,x,c),T(org~k,dst~j) → A(loc~j,x,c),T(org~k,dst~j) @ (1 * 1.1) / 500

pre-Kappa syntax: Location Matrices

...

#Location matrices

%locm:

TM	i	j	k
i	0	0.5	1.5
j	2.0	0	1.8
k	1.0	1.1	0

#A diffusions

%expand-rule: TM A(x,c),T() → A(%,x,c),T() @ 1

gives:

A(loc~i,x,c),T(org~i,dst~j) → A(loc~j,x,c),T(org~i,dst~j) @ (1 * 0.5) / 100

A(loc~i,x,c),T(org~i,dst~k) → A(loc~k,x,c),T(org~i,dst~k) @ (1 * 1.5) / 100

A(loc~j,x,c),T(org~j,dst~i) → A(loc~i,x,c),T(org~j,dst~i) @ (1 * 2.0) / 1000

A(loc~j,x,c),T(org~j,dst~k) → A(loc~k,x,c),T(org~j,dst~k) @ (1 * 1.8) / 1000

A(loc~k,x,c),T(org~k,dst~i) → A(loc~i,x,c),T(org~k,dst~i) @ (1 * 1.0) / 500

A(loc~k,x,c),T(org~k,dst~j) → A(loc~j,x,c),T(org~k,dst~j) @ (1 * 1.1) / 500

pre-Kappa syntax: Location Matrices

...
#Location matrices
%locm:

TM	i	j	k
i	0	0.5	1.5
j	2.0	0	1.8
k	1.0	1.1	0

#A diffusions
%expand-rule: TM A(x,c),T() → A(%,x,c),T() @ 1

gives:

A(loc~i,x,c),T(org~i,dst~j) → A(loc~j,x,c),T(org~i,dst~j) @ (1 * 0.5) / 100

A(loc~i,x,c),T(org~i,dst~k) → A(loc~k,x,c),T(org~i,dst~k) @ (1 * 1.5) / 100

A(loc~j,x,c),T(org~j,dst~i) → A(loc~i,x,c),T(org~j,dst~i) @ (1 * 2.0) / 1000

A(loc~j,x,c),T(org~j,dst~k) → A(loc~k,x,c),T(org~j,dst~k) @ (1 * 1.8) / 1000

A(loc~k,x,c),T(org~k,dst~i) → A(loc~i,x,c),T(org~k,dst~i) @ (1 * 1.0) / 500

A(loc~k,x,c),T(org~k,dst~j) → A(loc~j,x,c),T(org~k,dst~j) @ (1 * 1.1) / 500

pre-Kappa syntax: Location Matrices

...

#Location matrices

%locm:

TM	i	j	k
i	0	0.5	1.5
j	2.0	0	1.8
k	1.0	1.1	0

#Observing transporters

%expand-obs: TM 'Transporter(%org,%dst)' T()

gives:

%obs: 'Transporter(i,j)' T(org~i,dst~j)

%obs: 'Transporter(i,k)' T(org~i,dst~k)

%obs: 'Transporter(j,i)' T(org~j,dst~i)

%obs: 'Transporter(j,k)' T(org~j,dst~k)

%obs: 'Transporter(k,i)' T(org~k,dst~i)

%obs: 'Transporter(k,j)' T(org~k,dst~j)

pre-Kappa syntax: Location Matrices

...

#Location matrices

%locm:

TM	i	j	k
i	0	0.5	1.5
j	2.0	0	1.8
k	1.0	1.1	0

#Observing transporters

%expand-obs: TM 'Transporter(%org,%dst)' T()

gives:

%obs: 'Transporter(i,j)' T(org~i,dst~j)

%obs: 'Transporter(i,k)' T(org~i,dst~k)

%obs: 'Transporter(j,i)' T(org~j,dst~i)

%obs: 'Transporter(j,k)' T(org~j,dst~k)

%obs: 'Transporter(k,i)' T(org~k,dst~i)

%obs: 'Transporter(k,j)' T(org~k,dst~j)

pre-Kappa syntax: Chains

#Rules
'Expression' S(x!1),V(p!1,n!2),V(p!2,n!3),...,V(p!10,n) $\rightarrow$ Z() @ [inf]

gives:
'Expression' S(x!1),V(p!1,n!2),V(p!2,n!3),V(p!3,n!4), \
V(p!4,n!5),V(p!5,n!6),V(p!6,n!7),V(p!7,n!8),V(p!8,n!9), \
V(p!9,n!10),V(p!10,n) $\rightarrow$ Z() @ [inf]

pre-Kappa syntax: Chains

#Rules

'Expression' S(x!1),V(p!1,n!2),V(p!2,n!3),...,V(p!10,n) $\rightarrow$ Z() @ [inf]

gives:

'Expression' S(x!1),V(p!1,n!2),V(p!2,n!3),V(p!3,n!4), \
V(p!4,n!5),V(p!5,n!6),V(p!6,n!7),V(p!7,n!8),V(p!8,n!9), \
V(p!9,n!10),V(p!10,n) $\rightarrow$ Z() @ [inf]