**Stochastic Simulation**

- In explicit state simulation, there is one transition per reaction/rule, and each transition has a numerical weight (number of instances, multiplied by a rate constant). Updating weights is not a matter of doing some arithmetic.

- In implicit state simulation, there is one transition per instance. Weight = rate constant.

*Must update transitions, not weights:*

  - Add new transition.
  - Pick a transition.
  - Remove instance transition.

  [Conceptually, two phases: negative & positive update.]

---

**Multi-sums**

1. **Multi-sums (1)**
   - Recall the notion of pushout:
     
     ![Diagram](image)

     Construct $B \xrightarrow{D} C$ [st. the square commutes]

     Such that for any $B \xrightarrow{E} C$, there is a unique $D \xrightarrow{E}$ such that the triangle $B \xrightarrow{D} C \xrightarrow{E}$ commutes.

2. **Multi-sums (2)**
   - If $A$ is an initial object, i.e., there is exactly one arrow from $A$ to $B$ for all objects $B$.
   - A span $B \xrightarrow{E} C$ is essentially just a choice of two objects, $B$ and $C$, and the pushout is known as the sum, or co-product, written $B + C$.

3. **Multi-sums (3)**
   - Given $B$ and $C$, it is sometimes the case that there is not a unique object $B + C$ but a family of objects $D_i$:

     ![Diagram](image)

     Given $B, C$, we construct a family of co-spans such that any $B \xrightarrow{E} C$ factors uniquely through exactly one member of the family:

     ![Diagram](image)

4. **Multi-sums (4)**
   - The family of co-spans $B \xrightarrow{E} C$ is called the multi-sum (or multi-co-product) of $B$ and $C$.
Examples

- In the category of sets and total functions, the co-product of B and C is the disjoint union B ∪ C.

- Similarly for the categories of simple graphs or finite graphs and homomorphisms.

- In the category of sets and injective functions, the co-product breaks down:

5 GLUINGS AND OVERLAPS

- In the category of finite graphs and isomorphisms, any pair of objects, A and B, has a multi-sum.

- These describe the different ways that A and B can be 'glued together' to form a bigger graph.

- Any such gluing automatically induces an overlap (cf. intersection of sub-half-graphs) by taking the pull-back:

6 ACTIVATION (s)

- Activation is a relation between nodes.
- Given $r_i := \frac{p_i}{\ell_i}$ and $r_j := \frac{p_j}{\ell_j}$.

- We first construct the multi-sum $A \cup \ell_i, \ell_j$:

- Then we construct the overlap of each $G_i$:

7 'TYPES OF MATCHING PAIRS'

- Each member $G_i$ of the multi-sum of A and B defines one way that a pair of matchings — one from A, the other from B, can jointly map into some graph G:

- We want to say that $r_i$ activates $r_j$ iff for some overlap $O_i$, the further pull-back satisfies:

- This means that $O_i$ was modified in some way by the action of $r_i$.

- So a new instance of $r_j$ has been created...
Activation:

- another way to say this:
  if the error $O_i'$ is an isomorphism,
  
  This means that $O_i'$ is contained in $P_i$, the preserved region of $v_i$.
  
  $O_i'$ was not modified by $v_i$ (by def).

Inhibition:

- compute, as above, but for $L_1$ and $L_2$.

and then:

$O_i'$ is $O_i'$ is contained in $P_i$, and will not be modified by $v_i$.

$O_i'$ is $O_i'$ is contained in $P_i$, and will not be modified by $v_i$.

$O_i'$ inhibits $v_i$ if, for some $i$, $O_i' \neq O_i$.

Implicit State Simulation:

Given rules $v_1 \ldots v_n$ where

1. pick a rule $v_i$ and a matching $m_i: L_i \rightarrow G$
2. for all $j,k$, consider all factorizations $m_{ijk}: L_{ijk} \rightarrow G$
3. rewrite $G$ to $G'$ using $m_i$
4. for all $j,k$, consider all factorizations $m_{ijk}: L_{ijk} \rightarrow G'$ and all all induced $m_{ijk}: L_j \rightarrow G'$

and remove all induced $m_{ijk}: L_j \rightarrow G'$

$G_{ij} = \frac{R_i}{L_i} \frac{L_j}{G_{ijk}}$

$G_{ij}^+ = \frac{R_i}{L_i} \frac{L_j}{G_{ijk}^+}$

$G_{ij}^- = \frac{L_i}{L_j} \frac{G_{ijk}}{G_{ijk}^-}$

(Ik ranges over the family.)

N.B.: activation and inhibition are just relations between rules; but if we know that particular $G_i^+$ (or $G_i^-$) for which $O_i' \neq O_i$:

- any matching that factors thru $G_i^+$ (resp. $G_i^-$) will be created/deleted by the action of $v_i$.

- this is the basis of how we 'update weights' for graph-based/implicit state simulation.

ISS (1):

- as a first optimization, keep only those members of the $G_i^+ / G_i^-$ families that have a productive/correction-productive overlap; i.e.

  $O_i' \neq O_j^+$, $O_i' \neq O_j^-$

  *if $v_i$ does not activate $v_j$, there will be no productive overlap.

  *if $v_i$ does not inhibit $v_j$, there will be no correction-productive overlap.
**Implicit State Multi-set Rewriting**

1. Suppose we have two reactions:
   - \( r_1: A \rightarrow B + C \)
   - \( r_2: B + C \rightarrow D \)

2. As rewriting rules, these are:
   - \( r_1: \)
   - \( r_2: \)

3. There is no inhibition between these rules; however, \( r_1 \) activates \( r_2 \):
   - The multiset of \( r_1 \)'s RHS and \( r_2 \)'s LHS is:

   \[
   \begin{align*}
   &B, C \\
   &B, C, C \\
   &B, B, C \\
   &B, B, C, C \\
   &B, C, B, C, C \\
   \end{align*}
   \]

   - The first three cases give rise to a non-empty PB:

4. **Positive Update**:
   - After firing \( r_1 \), we have a matching for \( r_1 \)'s RHS to an updated state \( M' \):

   \[
   \begin{align*}
   &B, C \\
   &B, C, C \\
   &B, B, C \\
   \end{align*}
   \]

   - Which we must now factorize into the three productive overlaps with the LHS of \( r_2 \)

   - The factorization into \( B, C \) is trivial:
     - just \( M' \) itself

   - The factorization into \( B, B, C \) can occur in \( M'(B)-1 \) different ways (the non-overlapping C can map anywhere [different to the choice of \( M' \)]:

   - Likewise, the factorization into \( B, B, C \) can occur in \( M'(B)-1 \) different ways

5. **Update (2)**:
   - The first factorization is an instance of \( r_2 \) that
     - adds the new \( B \) and the new \( C \)
   - The second class of factorizations use the new \( B \) and an old \( C \)
   - The third class uses an old \( B \) and a new \( C \)

6. **NB**:
   - Implicit state multi-set rewriting is strange:
     - Counting matchings is the same as counting the number of \( A, B, C, D \) explicitly [less]
     - in effect, it is a complicated way of doing exactly the same as explicit multi-set rewriting!

   - unless...

   - we don't count matchings!

   - instead, associate each matching with its own private timer.

   - 'next event' is now simply the fastest event; and

   - negative/positive update removes/adds the matchings/timers as required

   - This is exactly the 'next event' method of simulation [from lecture 3]
ISS: efficiency concerns

- Def of true-/ve update is elegant; but...

- This is a problem when $C^{-1/f}_{ij,k}$ has one or more connected components that are not all in the image of the arrow $L_i/R_i ightarrow C^{-1/f}_{ij,k}$.

- If all connected components of $C^{-1/f}_{ij,k}$ are in this image, then there is at most one way to factorize $m_i$.

- This is non-trivial (but not too hard) and depends on realizability of site graphs and the fact that sites are all distinct.

- If we consider matchings from each connected component of the LHS/RHS of a rule separately, all the $C^{-1/f}_{ij,k}$ graphs are connected too!

- We are always in the good case above...

---

CLASHES

- If rule $R_i$ has $n_e$ connected components in its LHS and $n_e$ in its RHS, we have $n_e$ independent matchings for its LHS and $n_e$ for its RHS.

- If all connected components are mapped to disjoint parts of $M$, all is okay.

- By distinct c.c.s to distinct c.c.s in event: distinct c.c.s to disjoint areas of a single c.c.

- But it can happen that the collection of independent matchings is not collectively a valid matching.

- Two matchings can target the same agent or site...

- We need to find a way to deal with these "false" events dynamically.

---

NULL EVENTS

- A "fake", or null, event is just a loop in the underlying Markov jump process... that takes time!

- If the sum of all true event weights is $T$ and the sum of all null event weights is $N$,

- The probability of looping once is $\frac{N}{N+T} = q$ where $q = \frac{1}{q}$.

- More generally, the total time to exit a state is $N$ time to loop a time times follow a true event

- "Erlang" distribution (like G):

$$p(e) = \sum_{n=0}^{\infty} e^n \frac{(N+T)^n}{n!}$$

$$= p(N+T)e^{-(N+T)e} \sum_{n=0}^{\infty} \frac{e^n}{n!}$$

$$= T e^{-(N+T)e} e^{N}$$

$$= T \cdot e^{-(N+T)e} e^{N}$$

$$= T \cdot e^{-(N+T)e} e^{N}$$

as if the null events did not exist...
Null Events (2)

- We also need to verify that the exit probabilities are correct.

- If there are n true events, each with weight $T_i (1 \leq i \leq n)$ s.t.

\[
\sum_{i=1}^{n} T_i = T
\]

Then the probability of choosing a true event of type i is: \[\frac{T_i}{N+T}\]

So, the probability that the next true event is of type i is:

\[\frac{T_i}{N+T} / \frac{T}{N+T} = \frac{T_i}{T}\]

* This can also be derived by summing the geometric series (2)*

* In effect, a jump process w/o loops is the same as the underlying process w/ loops but with modified time advance.

AMBIGUOUS MOLECULARITY

- A binary rule may match a single connected component:

- Such an event has a different character than if it was matching two distinct connected components.

\[\Rightarrow\] the latter involves a collision — and so has a volume-dependent rate.

\[\Rightarrow\] the former is independent of the (system) volume...

\[\Rightarrow\] as if it were a unimolecular reaction with a first-order rate constant.

\[\Rightarrow\] This constant factors in the (small) 'reachin volume'-dependent that is implicit.

\[\Rightarrow\] rule should have its rate constants.

OVER-Sampling

- If you want to perform some action once per second (on average), the easiest solution is to use a clock ticking once per second...

\[\Rightarrow\] But could also use a 'clock' and 'roll a Coin' at each tick [null events...]

\[\Rightarrow\] This is a way to accumulate a second event that you wish to occur ten times per second.

\[\Rightarrow\] At each tick, do an 'a' with probability \(\frac{1}{10}\)

\[\Rightarrow\] and do a 'b' with probability \(\frac{9}{10}\).

OVER-Sampling (2)

- We can use this idea to deal with rules having two rate constants.

\[\Rightarrow\] a 'fast' clock for inter-molecular instances.

\[\Rightarrow\] Will lead to null events.

\[\Rightarrow\] if most instances are inter- [i.e., the system is in a 'percolated state'], null events will be rare.

\[\Rightarrow\] if most instances are inter-, most events will be null — not good?

- A context-specific strategy for the simulator.

\[\Rightarrow\] Maybe dynamically switch between over-sampling and other methods?

\[\Rightarrow\] or bound over-sampling dynamically?