LFCS

Laboratory for Foundations of Computer Science
Department of Computer Science - University of Edinburgh

Task Allocation in Monomorphic Ant Species

by

Chris Tofts

Task Allocation in Monomorphic Ant Species

LFCS Report Series **ECS-LFCS-91-144**

LFCS **March 1991**

Department of Computer Science
University of Edinburgh
The King's Buildings
Edinburgh EH9 3JZ

Task Allocation in Monomorphic Ant Species.

Chris Tofts
LFCS, Dept of Computer Science
University of Edinbirgh
Email:cmnt@uk.ac.ed.lfcs.

March 22, 1991

Abstract

The method whereby ant species with no physical differentiation allocate themselves to tasks within a colony is studied. We consider the various tasks as forming a production line, and demonstrate that various algorithms yield a correct arrangement of the individuals. The various models are expressed in the calculus WSCCS [Tof90a], to which we give a brief introduction.

1 Introduction.

Ant colony survival depends upon the succesful performance of a number of tasks. These include the feeding of larval stages, the maintenance of the nest structures, the collection of food and guarding the nest. Whilst it could be the case that each ant could perform a part of each of these tasks this is not observed [OW78, WH88, HW90]. Individuals have long-term preferences for particular tasks. So there is some division of labour within the nest. In some species this is achieved by physical suitability to a particular task (*morphological polyethism*) [HW90] within others the ants are all physically identical and yet must still manage to distribute themselves amongst the various tasks. The later form of organisation is usally referred to as *temporal polyethism* [Cal88,HW90]. Later we will see that the temporal differences observed do not need any temporal behaviour dependencies.

More basically we can ask what is a good algorithm for task allocation. We wish to achieve a situation where:

1. if possible all the required work is done;
2. no individual is overloaded;

thus a task allocation algorithm will be correct if it arranges the objects carrying out the work in proportion to the amount of work within a task. We will demonstrate that in many circumstances our method can achieve this result. We will use the calculus WSCCS to demonstrate these results, thus a brief introduction to that reasoning system is included. A more extensive introduction to WSCCS can be found in [Tof90a] and an extended example and its biological consequences can be found in [Tof90b, THF91].

2 The Language WSCCS.

Our language WSCCS is an extension of Milner's SCCS [Mil83] a language for describing synchronous concurrent systems. To define our language we presuppose an abelian group Act of atomic action symbols with identity $\mathbf{1}$ and the inverse of a being $\overline{a}$. As in SCCS, the complements a and $\overline{a}$ form the basis of communication. We also take a set of weights $\mathcal{W}$, denoted by w_i, which are the positive natural numbers $\mathcal{P}$ augmented with a set of infinite objects ω^k (with $k > 0$), with the following multiplication and addition rules (assuming $k > k'$):

$$n + w^k = w^k = w^k + n \quad w^k + w^{k'} = w^k = w^{k'} + w^k$$
$$n * w^k = w^k = w^k * n \quad w^k * w^{k'} = w^{k+k'} = w^{k'} * w^k,$$

and a set of process variables Var.

The collection of WSCCS expressions ranged over by E is defined by the following BNF expression, where $a \in Act$, $X \in Var$, $w_i \in \mathcal{W}$, S ranging over renaming functions, those $S : Act \longrightarrow Act$ such that $S(\mathbf{1}) = \mathbf{1}$ and $\overline{S(a)} = S(\overline{a})$, action sets $A \subseteq Act$, with $\mathbf{1} \in A$, and arbitrary *finite* indexing sets I:

$$E ::= X \mid a.E \mid \sum\{w_i E_i | i \in I\} \mid E \times E \mid E\lceil A \mid \Theta(E) \mid E[S] \mid \mu_i \tilde{x} \tilde{E}.$$

We let Pr denote the set of closed expressions, and add $\mathbf{0}$ to our syntax, which is defined by $\mathbf{0} \stackrel{def}{=} \sum\{w_i E_i | i \in \emptyset\}$.

The informal interpretation of our operators is as follows:

- $\mathbf{0}$ a process which cannot proceed;
- X the process bound to the variable X;
- $a.E$ a process which can perform the action a whereby becoming the process described by E;
- $\sum\{w_i E_i | i \in I\}$ the *weighted* choice between the processes E_i, the weight of the outcome E_i being determined by w_i. We think in terms of repeated experiments on this process and we expect to see over a large number of experiments the process E_i being chosen with a relative frequency of $\frac{w_i}{\sum_{i \in I} w_i}$.
- $E \times F$ the synchronous parallel composition of the two processes E and F. At each step each process must perform an action, the composition performing the composition (in Act) of the individual actions;
- $E\lceil A$ represents a process where we only permit actions in the set A. This operator is used to enforce communication and bound the scope of actions;
- $\Theta(E)$ represents taking the prioritised parts of the process E only.
- $E[S]$ represents the process E relabelled by the function S;

- $\mu_i \tilde{x} \tilde{E}$ represents the solution x_i taken from solutions to the mutually recursive equations $\tilde{x} = \tilde{E}$.

Often we shall omit the dot when applying prefix operators; also we drop trailing **0**, and will use a binary plus instead of the two (or more) element indexed sum, thus writing $\sum\{1_1 a.\mathbf{0},\ \ 2_2 b.\mathbf{0} | i \in \{1,2\}\}$ as $1.a\ +\ 2.b$. Finally we allow ourselves to specify processes definitionally, by providing recursive definitions of processes. For example, we write $A \stackrel{def}{=} a.A$ rather than $\mu x.ax$.

2.1 The Semantics of WSCCS.

In this section we define the operational semantics of WSCCS. The semantics is transition based, structurally presented in the style of [Plo81], defining the actions that a process can perform and the weight with which a state can be reached. In Figure 1 we present the operational rules of WSCCS. They are presented in a natural deduction style. The transitional semantics of WSCCS is given by the least relation $\longrightarrow \subseteq WSCCS \times Act \times WSCCS$ and the least multi-relation $\longmapsto \subseteq bag(WSCCS \times \mathcal{W} \times WSCCS)$ [1], which are written $E \stackrel{a}{\longrightarrow} F$ and $E \stackrel{w}{\longmapsto} F$ respectively, satisfying the rules laid out in Figure 1. These rules respect the informal description of the operators given earlier. The reason that processes are multi-related by weight is that we may specify more than one way to choose the same process with the same weight and we have to retain all the copies. For example, the process

$$1P + 1P + 1Q$$

can evolve to the process P with cummulative weight 2, so that we have to retain both evolutions.

The predicate $does_A(E)$ is well defined since we have only permitted finitely branching choice expressions. The action of the permission operator is to prune from the choice tree those processes that can no longer perform any action.

2.1.1 Direct Bisimulation.

Our bisimulations will be based on the accumulation technique of Larsen and Skou [LS89]. We start by defining accumulations of evolutions for both types of transition.

Definition 2.1 *Let S be a set of processes then:*

- $P \stackrel{w}{\longmapsto} S$ *with* $w = \sum\{w_i | P \stackrel{w_i}{\longmapsto} Q$ *for some* $Q \in S\}$; [2]
- $P \stackrel{a}{\longrightarrow} S$ *iff there exists* $Q \in S$ *and* $P \stackrel{a}{\longrightarrow} Q$.

[1] Where $\longmapsto \subseteq bag(WSCCS \times \mathcal{W} \times WSCCS)$ is the bag whose elements are those of the set $WSCCS \times \mathcal{W} \times WSCCS$, with the usual notion of bag.

[2] *Remembering this is a multi-relation so some of the Q and w_i may be the same process and value. We take all occurences of processes in S and add together all the weight arrows leading to them.*

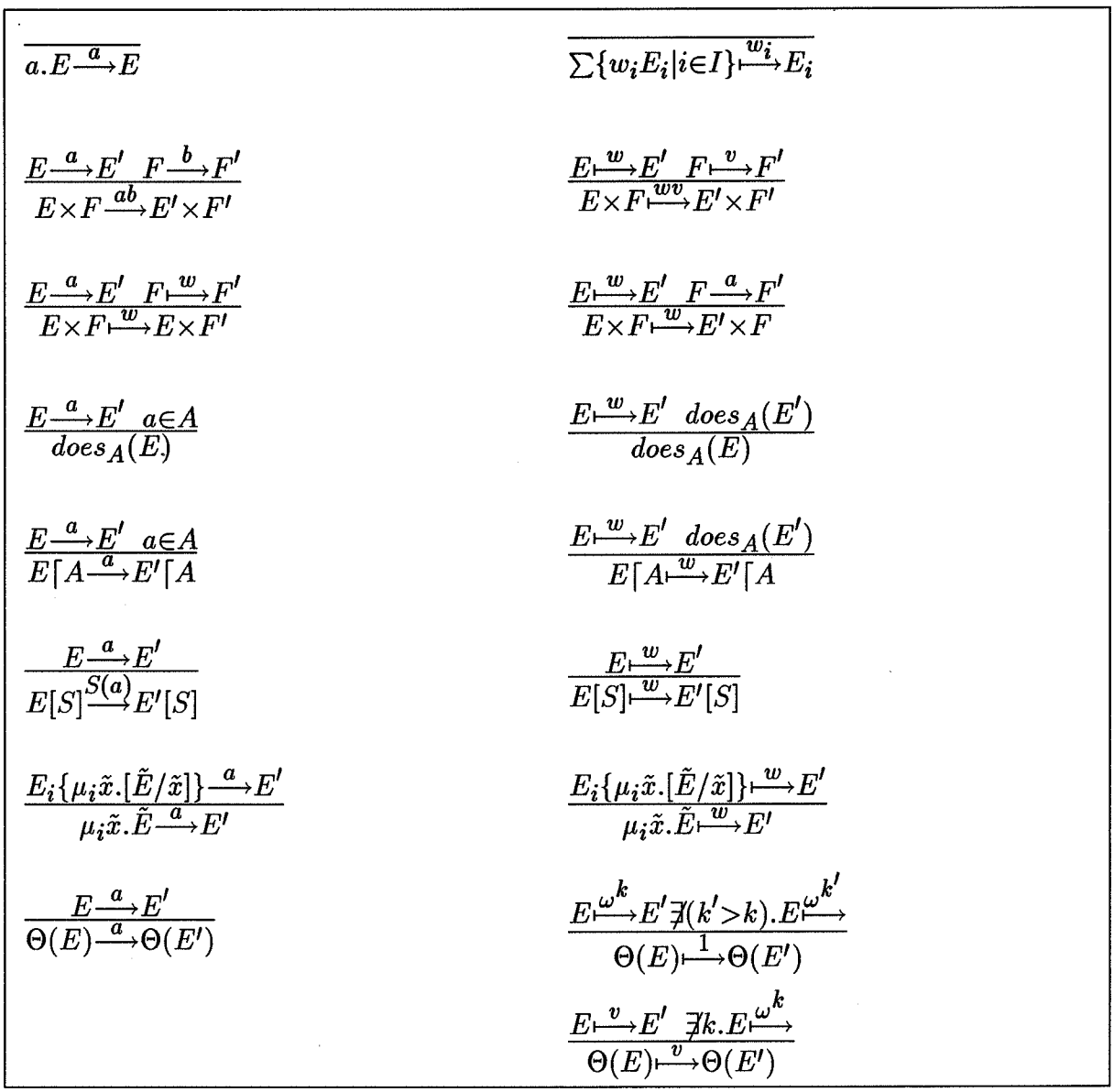

Figure 1: Operational Rules for WSCCS.

We define a form of bisimulation that identifies two processes if the total weight of evolving into any equivalent states is the same. This is not quite the indentification we wish to make, but we will make such an identification later.

Definition 2.2 *An equivalence relation $R \subseteq Pr \times Pr$[3] is a* direct bisimulation *if $(P,Q) \in R$ implies for all $S \in Pr/R$ that:*

- *for all $w \in \mathcal{W}$, $P \stackrel{w}{\longmapsto} S$ iff $Q \stackrel{w}{\longmapsto} S$;*
- *for all $a \in Act$, $P \stackrel{a}{\longrightarrow} S$ iff $Q \stackrel{a}{\longrightarrow} S$.*

Two processes are direct bisimulation equivalent, *written $P \stackrel{d}{\sim} Q$, if there exists a direct bisimulation R between them.*

Definition 2.3

$$\stackrel{d}{\sim} \equiv \bigcup\{R \mid R \text{ is a direct bisimulation }\}.$$

That $\stackrel{d}{\sim}$ is an equivalence follows immediately from it being a union of equivalences.

Lemma 2.4 *Let P and Q be processes such that $P \stackrel{d}{\sim} Q$. Then for all action sets A, $does_A(P)$ iff $does_A(Q)$.*

Proposition 2.5 *Direct equivalence is substitutive for finite processes. Thus, given $P \stackrel{d}{\sim} Q$ and $P_i \stackrel{d}{\sim} Q_i$ for all $i \in I$ then:*

1. $a.P \stackrel{d}{\sim} a.Q$;
2. $\Sigma_{i\in I} w_i P_i \stackrel{d}{\sim} \Sigma_{i\in I} w_i Q_i$;
3. $P \times E \stackrel{d}{\sim} Q \times E$;
4. $P\lceil A \stackrel{d}{\sim} Q\lceil A$;
5. $P[S] \stackrel{d}{\sim} Q[S]$.

We proceed by the usual technique of pointwise extension to define our equivalence for finite state processes.

Definition 2.6 *Let $\tilde{E}$ and $\tilde{F}$ be expressions containing variables at most $\tilde{X}$. Then we will say $\tilde{E} \stackrel{d}{\sim} \tilde{F}$ if for all process sets $\tilde{P}$, $\tilde{E}\{\tilde{P}/\tilde{X}\} \stackrel{d}{\sim} \tilde{F}\{\tilde{P}/\tilde{X}\}$.*

Proposition 2.7 *If $\tilde{E} \stackrel{d}{\sim} \tilde{F}$ then $\mu_i \tilde{X}.\tilde{E} \stackrel{d}{\sim} \mu_i \tilde{X}.\tilde{F}$.*

[3] *We denote the equivalence class of a process P with respect to R by $[P]_R$. When it is clear from the context to which equivalence we are refering, we will omit the subscript.*

2.1.2 Relative Bisimulation.

Unfortunately, the congruence given by direct bisimulation is too strong; it does not capture our notion of relative frequency, but captures total frequency. Since we would like to be able to equate processes such as,

$$2P + 3Q \text{ and } 4P + 6Q,$$

we need to weaken our notion of equality. The basic idea is that in order to show two processes equivalent, for each pair of equivalent states we can choose a *constant* factor such that the total weight of equivalent immediate derivatives is related by multiplication by that factor. If we can do this for all potentially equivalent states then we will say that the processes are the same in terms of relative frequency. Since the constant factor may well need to be a rational (and we wish to keep our numbers as simple as possible) we will actually use two constants in comparing relative frequency. This allows us to use a symmetrical definition.

Definition 2.8 *We say an equivalence relation $R \subseteq Pr \times Pr$ is a* relative bisimulation *if $(P, Q) \in R$ implies that:*

1. *there are $c_1, c_2 \in \mathcal{P}$ such that for all $S \in Pr/R$ and for all $w, v \in \mathcal{W}$, $P \stackrel{w}{\longmapsto} S$ iff $Q \stackrel{v}{\longmapsto} S$ and $c_1 w = c_2 v$;*
2. *for all $S \in Pr/R$ and for all $a \in Act$, $P \stackrel{a}{\longrightarrow} S$ iff $Q \stackrel{a}{\longrightarrow} S$.*

Two processes are relative bisimulation equivalent, *written $P \stackrel{r}{\sim} Q$ if there exists a relative bisimulation R between them.*

We have chosen to use multiplication by a constant rather than division as this permits us to stay within the natural numbers. We could have normalized so that the total weight actions of any state is 1, and then we would have had an equivalence that is identical to that of stratified bisimulation [SST89,GSST90].

Definition 2.9

$$\stackrel{r}{\sim} \equiv \bigcup\{R \mid R \text{ is a relative bisimulation}\}.$$

Proposition 2.10 *Let P and Q be processes such that $P \stackrel{d}{\sim} Q$, then $P \stackrel{r}{\sim} Q$.*

Definition 2.11 *Let $\tilde{E}$ and $\tilde{F}$ be expressions containing variables at most $\tilde{X}$. Then we will say $\tilde{E} \stackrel{r}{\sim} \tilde{F}$ if for all process sets $\tilde{P}$, $\tilde{E}\{\tilde{P}/\tilde{X}\} \stackrel{r}{\sim} \tilde{F}\{\tilde{P}/\tilde{X}\}$.*

Proposition 2.12 *$\stackrel{r}{\sim}$ is a congruence for finite and finite state processes.*

We would like a notion of equivalence that permits us to disregard the structure of the choices and just look at the total chance of reaching any particular state. This is known *not* to produce a congruence [SST89], but is a useful notion of equivalence.

Definition 2.13 *We define an abstract notion of evolution as follows;*

$P \xrightarrow{a[w]} P'$ *iff* $P \stackrel{w_1}{\longmapsto} \ldots \stackrel{w_n}{\longmapsto} \xrightarrow{a} P'$ *with* $w = \prod w_i$.

In order to define an equivalence which uses such transitions we need a notion of accumulation.

Definition 2.14 *Let S be a set of processes then:*

$P \xrightarrow{a[w]} S$ *iff* $w = \sum\{w_i | P \xrightarrow{a[w_i]} Q$ *for some* $Q \in S\}$; [4]

We can now define an equivalence that ignores the choice structure but not the choice values.

Definition 2.15 *We say an equivalence relation $R \subseteq Pr \times Pr$ is an* abstract bisimulation *if $(P, Q) \in R$ implies that:*

there are $c_1, c_2 \in \mathcal{P}$ such that for all $S \in Pr/R$ and for all $w, v \in \mathcal{W}$, $P \xrightarrow{a[w]} S$ iff $Q \xrightarrow{a[v]} S$ and $c_1 w = c_2 v$.

Two processes are abstract bisimulation equivalent, *written $P \overset{a}{\sim} Q$ if there exists a abstract bisimulation R between them.*

2.2 Equational Characterisation of WSCCS.

We present some equational laws over WSCCS processes in Figure 2, these form a sound and complete equational system over the finite processes in WSCCS. We shall write $p = q$ for $p \overset{\tau}{\sim} q$.

Definition 2.16 *Let A be an action set then the predicate, $d_A(E)$, expressing the fact that E can perform an action in A, is defined recursively as follows:*

- *If $a \in A$ then $d_A(a.E)$;*
- *If there exists $i \in I$ with $d_A(E_i)$ then $d_A(\Sigma_{i \in I} w_i E_i)$.*

Definition 2.17 *Let W be a set of weights $\{w_i\}$ then $max_\omega(W)$ is the maximum power of ω occuring in W, or 1 if there is no ω occurence in W.*

The major difference when we extend our weight set to have many infinities is that the priority operator will now distribute over multiplication. The following equation now holds:

$$\Theta(P \times Q) = \Theta(P) \times \Theta(Q)$$

this permits much greater freedom in the use of priority and ensures that it more closely matches with our intuitions.

[4] *Remembering this is a multi-relation so some of the Q and w_i may be the same process and value. We take all occurences of processes in S and add together all the weight arrows leading to them.*

$$(\Sigma_1)\ \Sigma_{i\in I}w_iE_i = \Sigma_{j\in J}v_jE_j \quad \begin{cases}\text{there is a surjection } f : I \longmapsto J \text{ with}\\ v_j = \Sigma\{w_i | i \in I \wedge f(i) = j\},\\ \text{and for all } i \text{ with } f(i) = j \text{ then } E_i = E_j.\end{cases}$$

$$(Exp_1)\ a.E \times b.F = ab.(E \times F) \qquad (Exp_2)\ a.E \times \Sigma_{j\in J}v_jF_j = \Sigma_{j\in J}v_j(a.E \times F_j)$$

$$(Res_1)\ (a.E)\lceil A = \begin{cases}a.(E\lceil A) \text{ if } a \in A\\ \mathbf{0} \text{ otherwise.}\end{cases}$$

$$(Res_2)\ (\Sigma_{i\in I}w_iE_i)\lceil A = \Sigma_{j\in J}w_j(E_j\lceil A) \text{ where } J = \{i \in I | d_A(E_i)\}$$

$$(\Theta_1)\ \Theta(a.E) = a.\Theta(E)$$

$$(\Theta_2)\ \Theta(\Sigma_{i\in I}w_iE_i) = \begin{cases}\Sigma_{j\in J}1.\Theta(E_j) \text{ where } J = \{i \in I | w_i = \omega^{max_w(\{w_i\})}\} \text{ and}\\ J \neq \emptyset,\\ \Sigma_{i\in I}w_i\Theta(E_i) \text{ if } J = \emptyset\end{cases}$$

$$(Ren)\ \Sigma_{i\in I}w_iE_i = \Sigma_{i\in I}nw_iE_i \text{ where } n \in \mathcal{P}$$

Figure 2: Equational rules for WSCCS.

3 The Basic Model.

In order to realise a task allocation stratagem we consider that our ants are involved in some form of production line (Figure 3), similar to Milner's Jobshop [Mil90]. We assume there are units that are required to be worked on proceeding from one task zone to the next. The different tasks are numbered from 1 to t. The basic idea is that if an ant finds too little work entering into its task zone then it should consider moving up a task and similarly if too little work is being taken from the zone then it should consider moving down a task. In this initial description we are assuming that there is sufficient work for everyone and that we simply have to achieve an internal balance between the numbers allocated to each particular task. Later we will examine a situation were the work load for each task is specified by an external process.

So the basic algorithm that an individual employs is as follows:

- we assume that there are t different tasks arranged linearly;
- attempt to take a task from the right and give it to the left;
- if this succeeds then stay where you are and remember that you did find work;
- for each direction that failed to give or take work, increment the number of times that failure has occured;
- if the number failures for either direction exceeds a critical amount then choose to move in that direction with some probability.

In the following z ranges from 2 up to t. We can formalise the above algorithm for task allocation by the following process definition:

$$
\begin{aligned}
Tk_z(k_l,k_r) &= \omega.task_{z+1}\overline{task_z}.\mathbf{1}.Task_z(0,0) + 1.Tk'_z(k_l,k_r)\\
Tk'_z(k_l,k_r) &= (\omega.(1.task_{z+1}.Downdecide_z(k_l,k_r+1) +\\
&\qquad 1.\overline{task_z}.Updecide_z(k_l+1,k_r)) +\\
&\qquad 1.\mathbf{1}.Bothdecide_z(k_l+1,k_r+1))
\end{aligned}
$$

$$
Updecide_z(k_l,k_r) = \begin{cases} \mathbf{1}.Tk_z(k_l,k_r) & \text{if } k_r < k_{rc} \\ m.\mathbf{1}.Tk_{z+1}(0,0) + n.\mathbf{1}.Tk_z(0,0) & \text{otherwise.} \end{cases}
$$

$$
Downdecide_z(k_l,k_r) = \begin{cases} \mathbf{1}.Tk_z(k_l,k_r) & \text{if } k_l < k_{lc} \\ m'.\mathbf{1}.Tk_{z-1}(0,0) + n'.\mathbf{1}.Tk_z(k_l,k_r) & \text{otherwise.} \end{cases}
$$

$$
Bothdecide_z(k_l,k_r) = \begin{cases} mn'.\mathbf{1}.Tk_{z+1}(0,0) + m'n.\mathbf{1}.Tk_{z-1}(0,0) + \\ (mm'+nn').\mathbf{1}.Tk_z(0,0) & \text{if } k_l \geq k_{lc} \text{ and } k_r \geq k_{rc} \\ Updecide_z(k_l,k_r) & \text{if } k_r \geq k_{rc} \text{ and } k_l < k_{lc} \\ Downdecide_z(k_l,k_r) & \text{otherwise.} \end{cases}
$$

$$
\begin{aligned}
Tk_1(k_l,k_r) &= \omega.task_2.\mathbf{1}.Tk_1(0,0) + 1.\mathbf{1}.Updecide_1(k_l,k_r)\\
Updecide_1(k_l,k_r) &= \begin{cases} \mathbf{1}.Tk_1(k_l,k_r) & \text{if } k_r < k_{rc} \\ m.\mathbf{1}.Tk_2(0,0) + n.\mathbf{1}.Task_1(0,0) & \text{otherwise.} \end{cases}
\end{aligned}
$$

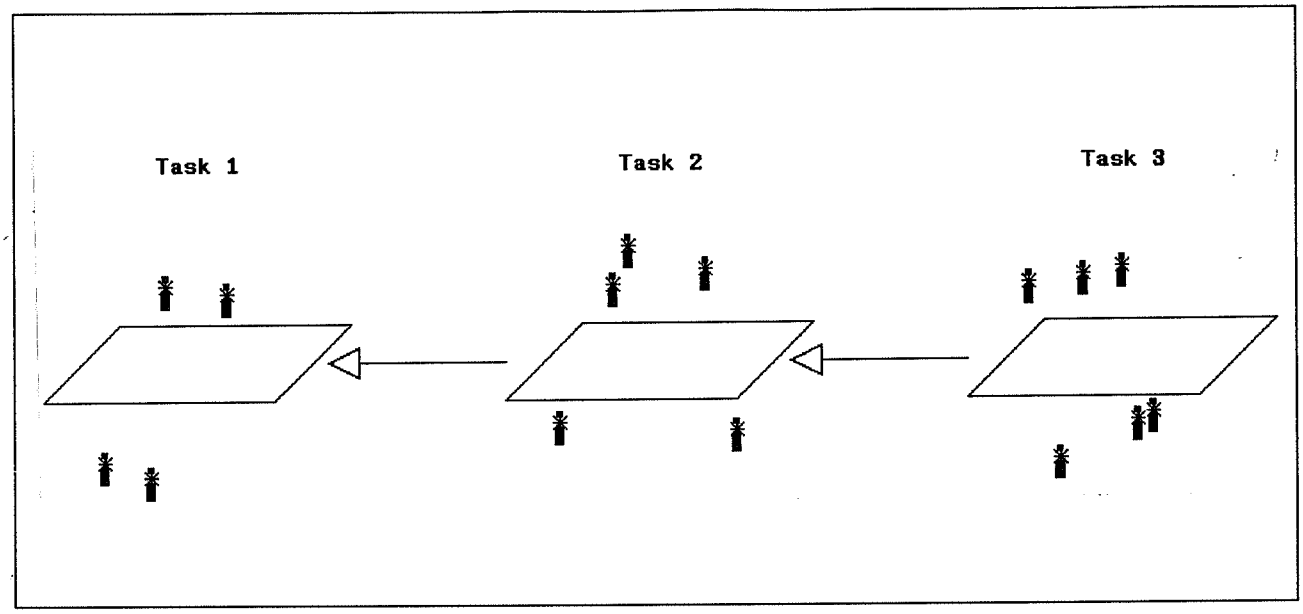

Figure 3: An ant production line.

$$Tk_t(k_l, k_r) = \omega.\overline{task_t}.\mathbf{1}.Tk_t(0,0) + \mathbf{1}.\mathbf{1}.Downdecide_t(k_l, k_r)$$

$$Downdecide_t(k_l, k_r) = \begin{cases} \mathbf{1}.Tk_t(k_l, k_r) & \text{if } k_l < k_{lc} \\ m'.\mathbf{1}.Tk_{t-1}(0,0) + n'.\mathbf{1}.Tk_z(k_l, k_r) & \text{otherwise.} \end{cases}$$

The constants in the above process description have the following intended meanings:

- k_{lc} and k_{rc} are the number of times we have to fail to find work in the directions left and right respectively before attempting to change task,
- $\frac{m}{m+n}$ is the probability of moving *up* a task given that there has been insufficient work from that direction,
- $\frac{m'}{m'+n'}$ is the probability of moving *down* a task, note that these probabilities need not be symmetric[5].

Take a vector x_i of numbers in the range 1 to t then our system is the following process:

$$Line = \Theta((\prod_{i=1}^{k} Tk_{x_i}(0,0))\lceil\{\mathbf{1}\})$$

Lemma 3.1 *If k is a multiple of t i.e. $k = k't$ then the following state is stable:*

[5]Indeed since the earlier tasks within a nest are safer from predation we might expect that these probabilities would be biased in favour of lower numbered tasks.

$$\Theta(((\prod_{i=1}^{k'} Tk_1(0,0)) \times (\prod_{i=1}^{k'} Tk_2(0,0)) \times \ldots \times (\prod_{i=1}^{k'} Tk_t(0,0)))\lceil\{\mathbf{1}\})$$

Proof: We start by calculating the expansion of the processes performing one of the tasks:

$$\begin{aligned}\prod_{i=1}^{k'} Tk_z(0,0) =& \omega^{k'}.task_{z+1}^{k'}\overline{task_z}^{k'}.\mathbf{1}.(\prod_{i=1}^{k'} Tk_z(0,0)) + \\ & \omega^{k'-1}.((\prod_{i=1}^{k'-1} task_{z+1}\overline{task_z}.\mathbf{1}.Tk_z(0,0)) \times Tk'_z(0,0)) + \\ & + \ldots + \\ & 1.(\prod_{i=1}^{k'} Tk'_z(0,0))\end{aligned}$$

Hence we can show that the process:

$$((\prod_{i=1}^{k'} Tk_1(0,0)) \times (\prod_{i=1}^{k'} Tk_2(0,0)) \times \ldots \times (\prod_{i=1}^{k'} Tk_t(0,0)))\lceil\{\mathbf{1}\}$$

is equivalent to:

$$\begin{aligned}&\omega^k.\mathbf{1}.\mathbf{1}.\ ((\prod_{i=1}^{k'} Tk_1(0,0)) \times (\prod_{i=1}^{k'} Tk_2(0,0)) \times \ldots \times (\prod_{i=1}^{k'} Tk_l(0,0)))\lceil\{\mathbf{1}\} \\ &+ \text{terms of lower order in } \omega.\end{aligned}$$

and hence our original process is equivalent (as required) to:

$$1.\mathbf{11}.\Theta(((\prod_{i=1}^{k'} Tk_1(0,0)) \times (\prod_{i=1}^{k'} Tk_2(0,0)) \times \ldots \times (\prod_{i=1}^{k'} Tk_t(0,0)))\lceil\{\mathbf{1}\})$$

We wish to establish[6] that any state which is not stable must eventually reach the stable state.

Lemma 3.2 *From an arbitrary set of values* $0 \leq x_i \leq t$ *and* u_i, d_i *substituted in the process:*

$$\Theta((\prod_{i=1}^{k} Tk_{x_i}(d_i,u_i))\lceil\{\mathbf{1}\})$$

there exists a finite *sequence of probabilistic transistions and* **1** *transitions leading to the stable state.*

[6] We use a simplification of the proof technique for synchronisation found in [Tof90b]

Proof: We demonstrate that the property holds for some particular state. This demonstrates how one may construct such a derivation path from an arbitrary state (we will enumerate each probabilistic transition with its probability, rather than its relative frequency, and assume that $k_{lc} = k_{rc}$):

$$\Theta(((\prod_{i=1}^{k'-1} Tk_1(0,0)) \times (\prod_{i=1}^{k'} Tk_2(0,0)) \times Tk_2(0,0) \times \ldots (\prod_{i=1}^{k'} Tk_t(0,0)))\lceil\{\mathbf{1}\})$$

$$\downarrow 1$$
$$\downarrow \mathbf{1}$$
$$\downarrow \mathbf{1}$$

$$\Theta(((\prod_{i=1}^{k'-1} Tk_1(0,0)) \times (\prod_{i=1}^{k'} Tk_2(0,0)) \times Tk_2(1,1) \times \ldots \times (\prod_{i=1}^{k'} Tk_t(0,0)))\lceil\{\mathbf{1}\})$$

$$\downarrow \frac{1}{k'+1}$$
$$\downarrow \mathbf{1}$$
$$\downarrow \mathbf{1}$$

$$\Theta(((\prod_{i=1}^{k'-1} Tk_1(0,0)) \times (\prod_{i=1}^{k'} Tk_2(0,0)) \times Tk_2(2,2) \times \ldots \times (\prod_{i=1}^{k'} Tk_t(0,0)))\lceil\{\mathbf{1}\})$$

$$\left.\begin{array}{l} \downarrow \frac{1}{k'+1} \\ \downarrow \mathbf{1} \\ \downarrow \mathbf{1} \end{array}\right\} k_{lc} - 3 \text{ times}$$

$$\Theta(((\prod_{i=1}^{k'-1} Tk_1(0,0)) \times (\prod_{i=1}^{k'} Tk_2(0,0)) \times Tk_2(k_{lc}-1, k_{lc}-1) \times$$
$$\ldots \times (\prod_{i=1}^{k'} Tk_t(0,0)))\lceil\{\mathbf{1}\})$$

$$\downarrow \frac{1}{k'+1}$$
$$\downarrow \mathbf{1}$$
$$\downarrow \frac{m'n}{(m+n)(m'+n')}$$
$$\downarrow \mathbf{1}$$

$$\Theta(((\prod_{i=1}^{k'} Tk_1(0,0)) \times (\prod_{i=1}^{k'} Tk_2(0,0)) \times \ldots \times (\prod_{i=1}^{k'} Tk_t(0,0)))\lceil\{\mathbf{1}\})$$

so with probability $(\frac{1}{k'+1})^{(k_{lc}-1)}(\frac{m'n}{(m+n)(m'+n')})$ we reach the stable state, on this transition path. Other examples are more complex versions of the above. Thus whenever the probability of deciding to move up or down is non-zero (m and m' non-zero) then we can always reach the stable state in a finite number of transitions.

Proposition 3.3 *Given any process Line with $k = k't$ then that process must eventually evolve to the process*

$$\Theta(((\prod_{i=1}^{k'} Tk_1(0,0)) \times (\prod_{i=1}^{k'} Tk_2(0,0)) \times \ldots \times (\prod_{i=1}^{k'} Tk_t(0,0)))\lceil\{\mathbf{1}\})$$

Proof: From Lemma 3.1 we have a stable state and from Lemma 3.2 we have a non-zero probability of reaching that state, so we must always eventually reach such a state.

In order to illustrate the above process 'homing' in on the correct solution we include a simulation (Figure 4) of 8 ants trying to allocate themselves to 4 tasks when all 8 start off working in the first task.

The question arises as to what happens when k is not an exact multiple of t, say $k = k't + w$ with $w < k'$. Then there will be w processes "wandering around". In the case where $w = 1$ if we repeat the analysis of Lemma 3.1 then we find that (when the extra Tk process is performing task 2 say):

$$\Theta(((\prod_{i=1}^{k'} Tk_1(0,0)) \times (\prod_{i=1}^{k'+1} Tk_2(0,0)) \times \ldots \times (\prod_{i=1}^{k'} Tk_t(0,0)))\lceil\{\mathbf{1}\})$$

the highest prioritised part of the process is the following:

$$\omega^{k't}.1.1.(Bothdecide_2(1,1) \times \mathbf{1}.\mathbf{1}.((\prod_{i=1}^{k'} Tk_1(0,0)) \times (\prod_{i=1}^{k'} Tk_2(0,0)) \times \ldots \times (\prod_{i=1}^{k'} Tk_t(0,0))))$$

given that k_{lc} and k_{rc} are greater than zero we arrive at the state:

$$\Theta((Tk_2(1,1) \times (\prod_{i=1}^{k'} Tk_1(0,0)) \times (\prod_{i=1}^{k'} Tk_2(0,0)) \times \ldots \times (\prod_{i=1}^{k'} Tk_t(0,0)))\lceil\{\mathbf{1}\})$$

So we have one process that will wander around whilst the others stay fixed in the correct arrangement.

3.1 Adding Task Difficulty.

So far we have assumed that all tasks have the same difficulty, that is to say we can achieve the same amount performing each task. If we assume that each task has a degree of ease e_z and redefine our Tk process as follows:

$$\begin{aligned}
Task_z(k_l, k_r) &= \omega.task_{z+1}^{e_z}\overline{task_z}^{e_z}.\mathbf{1}.Task(0,0) + 1.Task'_z(k_l, k_r) \\
Task'_z(k_l, k_r) &= (\omega.(1.task_{z+1}^{e_z}.Downdecide_z(k_l, k_r + 1) + \\
&\qquad 1.\overline{task_z}^{e_z}.Updecide_z(k_l + 1, k_r)) + \\
&\qquad 1.\mathbf{1}.Bothdecide_z(k_l + 1, k_r + 1))
\end{aligned}$$

$$Updecide_z(k_l, k_r) = \begin{cases} \mathbf{1}.Task_z(k_l, k_r) & \text{if } k_r < k_{rc} \\ m.\mathbf{1}.Task_{z+1}(0,0) + n.\mathbf{1}.Task_z(0,0) & \text{otherwise.} \end{cases}$$

$$Downdecide_z(k_l, k_r) = \begin{cases} \mathbf{1}.Task_z(k_l, k_r) & \text{if } k_l < k_{lc} \\ m'.\mathbf{1}.Task_{z-1}(0,0) + n'.\mathbf{1}.Task_z(k_l, k_r) & \text{otherwise.} \end{cases}$$

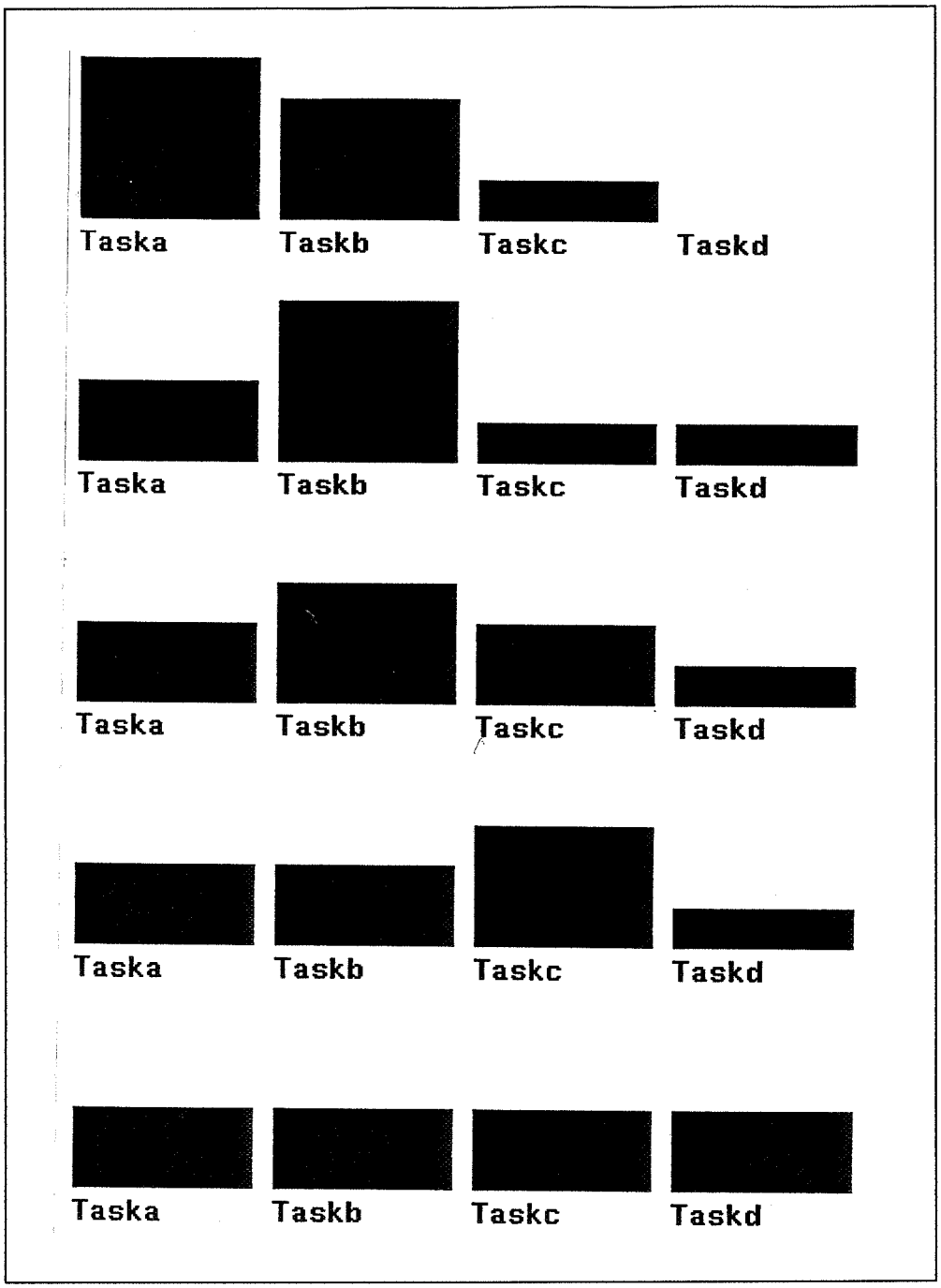

Figure 4: Eight ants evenly allocating themselves amongst 4 tasks.

$$Bothdecide_z(k_l,k_r) = \begin{cases} mn'.\mathbf{1}.Task_{z+1}(0,0) + m'n.\mathbf{1}.Task_{z-1}(0,0) + \\ (mm'+nn').\mathbf{1}.Task_z(0,0) \text{ if } k_l \geq k_{lc} \text{ and } k_r \geq k_{rc} \\ Updecide_z(k_l,k_r) \qquad \text{if } k_r \geq k_{rc} \text{ and } k_l < k_{lc} \\ Downdecide_z(k_l,k_r) \qquad \text{otherwise.} \end{cases}$$

$$Task_1(k_l,k_r) = \omega.task_2^{e_1}.\mathbf{1}.Task(0,0) + 1.\mathbf{1}.Updecide_1$$

$$Updecide_1(k_l,k_r) = \begin{cases} \mathbf{1}.Task_1(k_l,k_r) & \text{if } k_r < k_{rc} \\ m.\mathbf{1}.Task_2(0,0) + n.\mathbf{1}.Task_1(0,0) & \text{otherwise.} \end{cases}$$

$$Task_t(k_l,k_r) = \omega.\overline{task_t}^{e_t}.\mathbf{1}.Task(0,0) + 1.\mathbf{1}.Downdecide_t(k_l,k_r)$$

$$Downdecide_t(k_l,k_r) = \begin{cases} \mathbf{1}.Task_t(k_l,k_r) & \text{if } k_l < k_{lc} \\ m'.\mathbf{1}.Task_{t-1}(0,0) + n'.\mathbf{1}.Task_z(k_l,k_r) & \text{otherwise.} \end{cases}$$

Take a vector x_i of numbers in the range 1 to t then our system is the following process:

$$Line = \Theta((\prod_{i=1}^{k} Task_{x_i}(0,0))\lceil\{\mathbf{1}\})$$

Proposition 3.4 *if k is such that there exists $k_1, k_2, \ldots k_t$ with $k = k_1 + k_2 + \ldots + k_t$ and $k_1e_1 = k_2e_2 = \ldots = k_te_t$ then the following state is stable and moreover it must eventually be reached independently of starting state:*

$$\Theta(((\prod_{i=1}^{k_1} Task_1(0,0)) \times (\prod_{i=1}^{k_2} Task_2(0,0)) \times \ldots \times (\prod_{i=1}^{k_t} Task_t(0,0)))\lceil\{\mathbf{1}\})$$

Proof: Proceed as before, the stability is achieved by observing the expansion:

$$\begin{aligned} \prod_{i=1}^{k_z} Task_z(0,0) = {} & \omega^{k_z}.task_{z+1}^{e_zk_z}\overline{task_z}^{e_zk_z}.\mathbf{1}.(\prod_{i=1}^{k_z} Task_z(0,0)) + \\ & \omega^{k'-1}.((\prod_{i=1}^{k_t-1} ask_{z+1}\overline{task_z}.\mathbf{1}.Task_z(0,0)) \times Task'_z(0,0)) + \\ & + \ldots + \\ & 1.(\prod_{i=1}^{k_z} Task'_z(0,0)) \end{aligned}$$

And since $k_1e_1 = k_2e_2 = \ldots = k_te_t$ holds then the highest order term in ω is the following:

$$\omega^k.\mathbf{1}.1.((\prod_{i=1}^{k_1} Task_1(0,0)) \times (\prod_{i=1}^{k_2} Task_2(0,0)) \times \ldots \times (\prod_{i=1}^{k_t} Task_l(0,0)))\lceil\{\mathbf{1}\}$$

as we require.

Moreover from an arbitrary starting state there is a non-zero probability of reaching the stable state.

As an illustration of the system reaching its stable state we have a simulation (Figure 5) which demonstrates 7 ants allocating themselves to 3 tasks within which an individual can perform 1,2 and 4 units of work respectively.

One question that arises is what happens if the degree of difficulty of a task varies with time. Clearly the system can respond correctly to this consider a system at equilibrium with relative task eases of $e_1, e_2, \ldots, e_t$. So we start a system with the processes arranged for these difficulties but the difficulties are actually $e'_1, e'_2, \ldots, e_n$ clearly this will arrange itself correctly, after some period of time. So as long as the variability rate of the task difficulties is much lower than the rate at which equilibrium is established our system can keep assigning the correct number of individuals to each task. This is important in a biological system as the environment can greatly affect the difficulty of performance of such tasks as foraging.

4 External Work Levels.

We add a process for each task that will set the amount of work the processes in each task have to do at each turn. We define the producer process (producing work within a particular range) as follows:

$$Prod_z(r) = \Sigma_{i=0}^{i=r} 1.\overline{work}_z^i.\mathbf{1}.\textstyle\sum_{i \in Range} w_i.\ \mathbf{1}.Prod_z(i)$$

and we alter our initial version of the allocation system as follows:

$$\begin{aligned}
Wrk_z(k_l, k_r) &= \omega.work_z.\mathbf{1}.\mathbf{1}.Wrk_z(0,0) + 1.Tsk_z(k_l, k_r) \\
Tsk_z(k_l, k_r) &= \omega.task_{z+1}\overline{task_z}.\mathbf{1}.Wrk(0,0) + 1.Tsk'_z(k_l, k_r) \\
Tsk'_z(k_l, k_r) &= (\omega.(1.task_{z+1}.Downd_z(k_l, k_r + 1) + \\
&\qquad 1.\overline{task_z}.Upd_z(k_l + 1, k_r)) + \\
&\qquad 1.\mathbf{1}.Bothd_z(k_l + 1, k_r + 1))
\end{aligned}$$

$$Upd_z(k_l, k_r) = \begin{cases} \mathbf{1}.Wrk_z(k_l, k_r) & \text{if } k_r < k_{rc} \\ m.\mathbf{1}.Task_{z+1}(0,0) + n.\mathbf{1}.Wrk_z(0,0) & \text{otherwise.} \end{cases}$$

$$Downd_z(k_l, k_r) = \begin{cases} \mathbf{1}.Wrk_z(k_l, k_r) & \text{if } k_l < k_{lc} \\ m'.\mathbf{1}.Wrk_{z-1}(0,0) + n'.\mathbf{1}.Wrk_z(k_l, k_r) & \text{otherwise.} \end{cases}$$

$$Bothd_z(k_l, k_r) = \begin{cases} mn'.\mathbf{1}.Wrk_{z+1}(0,0) + m'n.\mathbf{1}.Wrk_{z-1}(0,0) + \\ (mm' + nn').\mathbf{1}.Wrk_z(0,0) & \text{if } k_l \geq k_{lc} \text{ and } k_r \geq k_{rc} \\ Upd_z(k_l, k_r) & \text{if } k_r \geq k_{rc} \text{ and } k_l < k_{lc} \\ Downd_z(k_l, k_r) & \text{otherwise.} \end{cases}$$

$$\begin{aligned}
Wrk_1(k_l, k_r) &= \omega.work_z.\mathbf{1}.\mathbf{1}.Wrk_1(0,0) + 1.Tsk_1(k_l, k_r) \\
Tsk_1(k_l, k_r) &= \omega.task_2.\mathbf{1}.Wrk_1(0,0) + 1.\mathbf{1}.Upd_1
\end{aligned}$$

$$Upd_1(k_l, k_r) = \begin{cases} \mathbf{1}.Wrk_1(k_l, k_r) & \text{if } k_r < k_{rc} \\ m.\mathbf{1}.Wrk_2(0,0) + n.\mathbf{1}.Wrk_1(k_l, k_r) & \text{otherwise.} \end{cases}$$

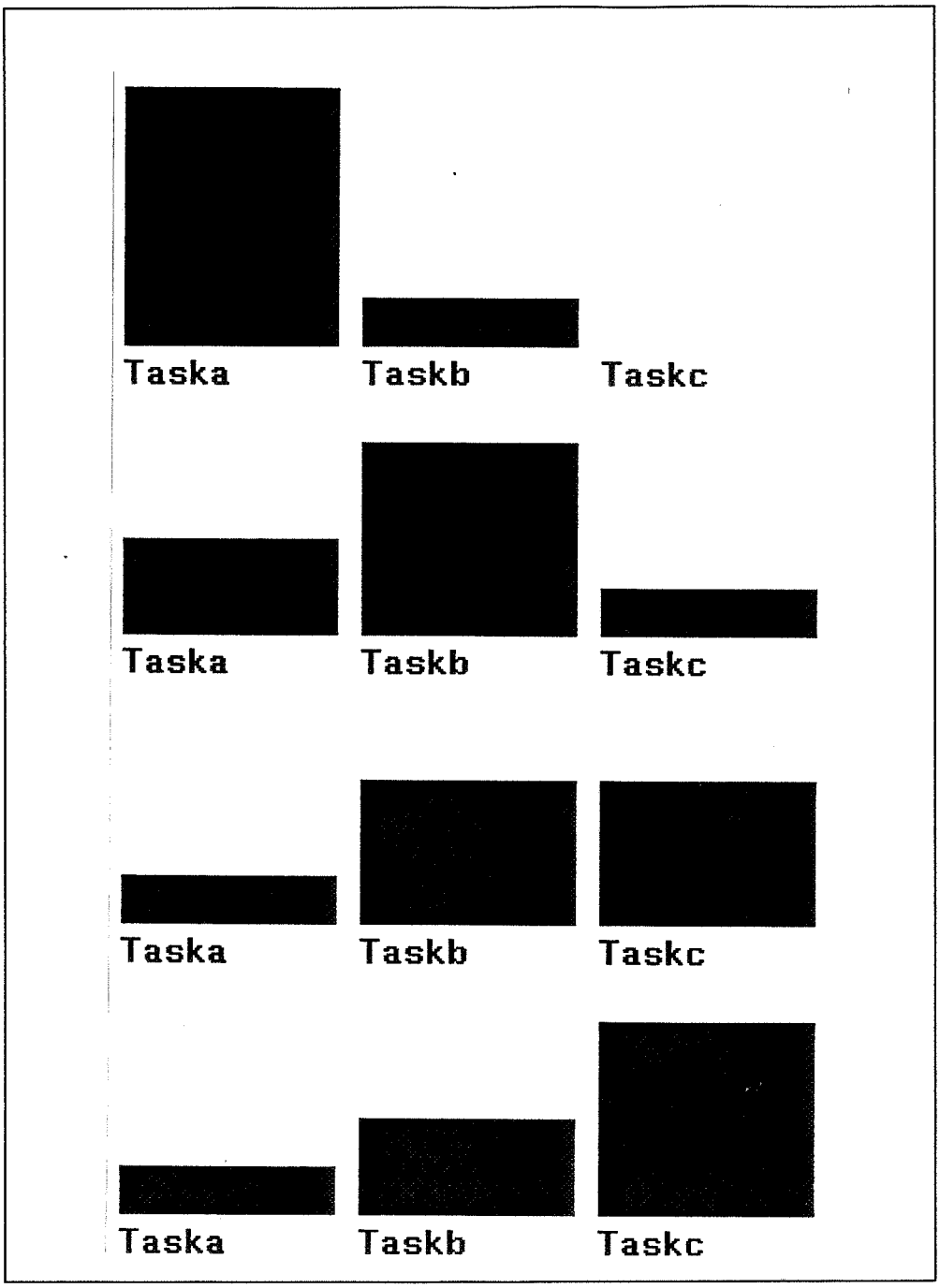

Figure 5: Seven ants allocating an uneven load amongst themselves.

$$Wrk_t(k_l, k_r) = \omega.work_z.\mathbf{1}.\mathbf{1}.Wrk_t(0,0) + 1.Tsk_t(k_l, k_r)$$
$$Tsk_t(k_l, k_r) = \omega.\overline{task_t}.\mathbf{1}.Wrk_t(0,0) + 1.\mathbf{1}.Downd_t(k_l, k_r)$$
$$Downd_t(k_l, k_r) = \begin{cases} \mathbf{1}.Wrk_t(k_l, k_r) & \text{if } k_l < k_{lc} \\ m'.\mathbf{1}.Wrk_{t-1}(0,0) + n'.\mathbf{1}.Wrk_z(k_l, k_r) & \text{otherwise.} \end{cases}$$

So our description of the task allocation system will be as follows (given a sequence x_i of values in the range $1 \ldots t$):

$$Ex = \Theta(((\prod_{i=1}^{k} Wrk_{x_i}(0,0)) \times Prod_1 \times \ldots \times Prod_t) \lceil \{\mathbf{1}\})$$

4.1 Fixed Work Levels.

If we take in our definition of the work load for each task a fixed amount produced at each turn; the case when the producer process is of the following form:

$$Prod_z(r) = \Sigma_{i=0}^{i=r} 1.\overline{work}_z^i.\mathbf{1}.\mathbf{1}.Prod_z(w_z)$$

then we have stability if the processes can arrange themselves so as to handle all the work available. Clearly this can only be acheived if the total amount of work is less than or equal to the number of processes available to handle it.

Proposition 4.1 *if $k = \sum w_z$ then there is a stable arrangement of the processes and moreover that arrangent must always be reached from any initial state.*

Proof: If we have w_z process perfoming task z (this is possible within the context of the total being k then we have the following expansion for process performing that task:

$$\begin{aligned} \prod_{i=1}^{w_z} Wrk_z(0,0) = & \omega^{w_z}.work^{w_z}.\mathbf{1}.\mathbf{1}.(\prod_{i=1}^{w_z} Wrk_z(0,0)) + \\ & \omega^{w_z-1}.work^{w_z-1}.(Task_z(0,0) \times (\prod_{i=1}^{w_z-1} Wrk_z(0,0))) \\ & + \ldots + \\ & \mathbf{1}.(\prod_{i=1}^{w_z} Task_z(0,0)) \end{aligned}$$

That this state can be reached in a finite number of steps follows from the allocation activities of the underlying $Task$ system.

We include two illustrations of this situation. In the first (Figure 6) there are three tasks with 1, 3 and 3 amounts of work to be done in each repectively. In the second illustration (Figure 7) there are four tasks with 1, 2, 2 and 3 amounts of work to be performed in each task respectively.

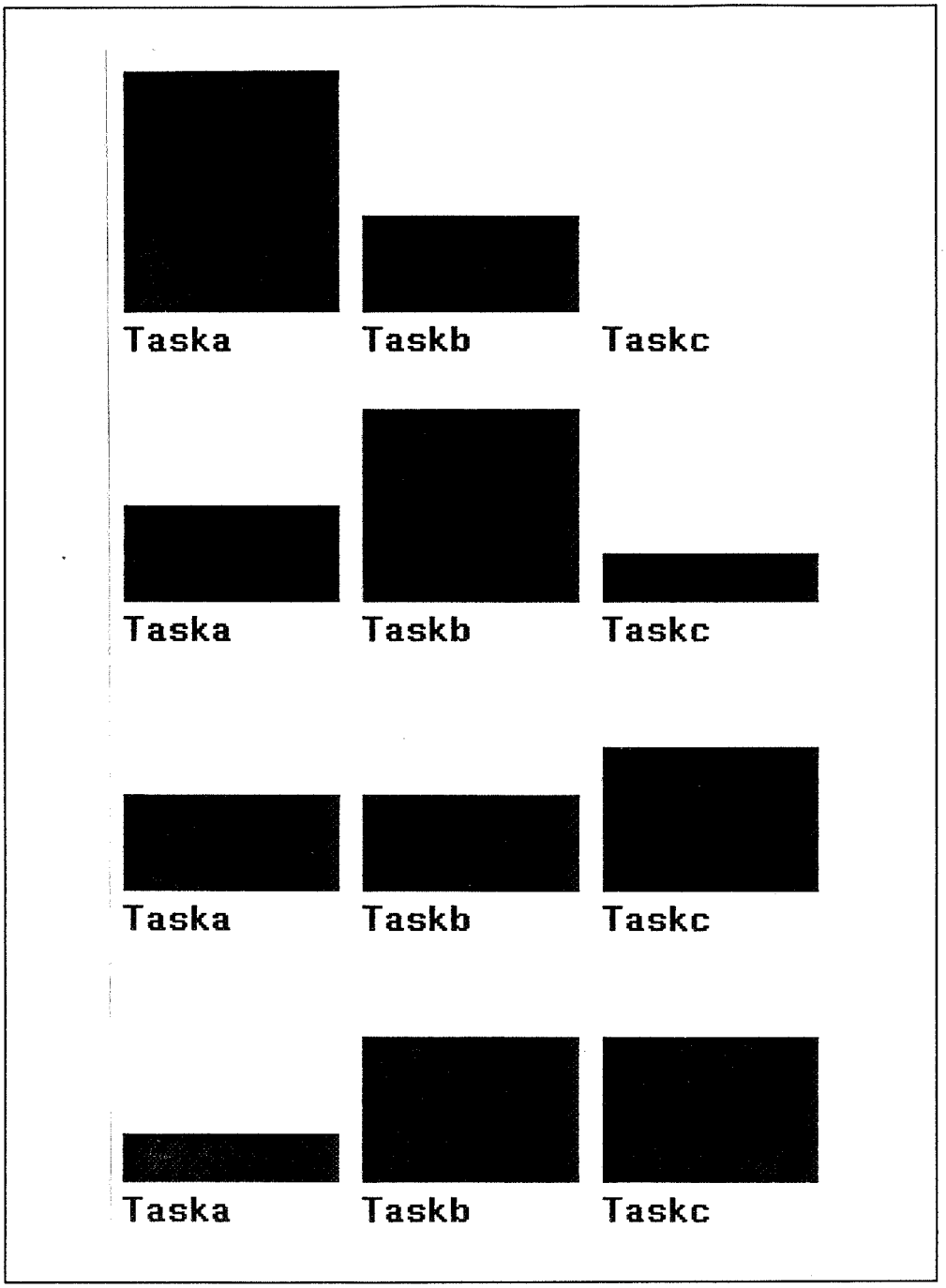

Figure 6: Seven ants with an external workload of 1,3,3 respectively.

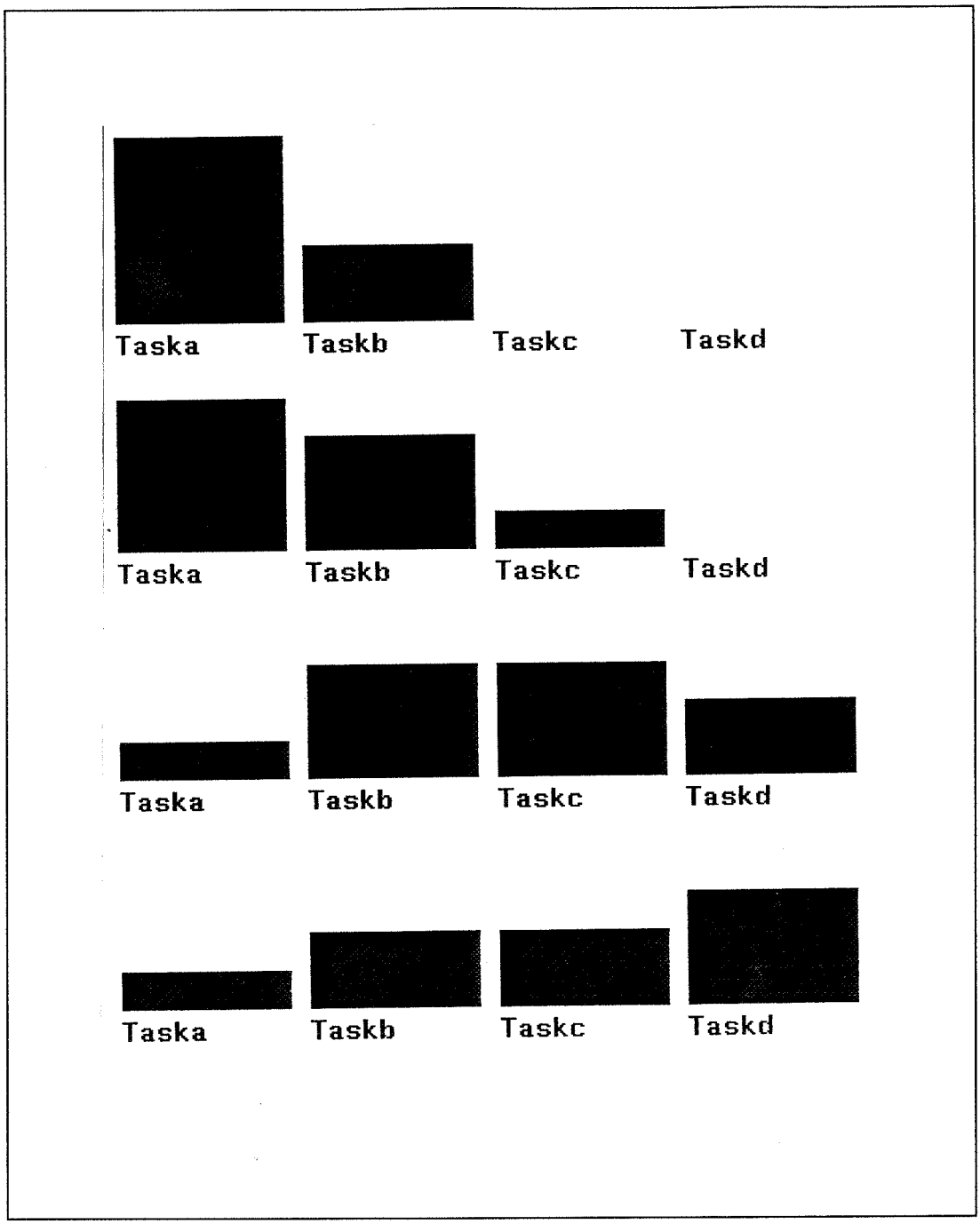

Figure 7: Eight ants with an external workload of 1,2,2,3 repectively.

4.2 Variable Work Levels.

When the producer does not produce a fixed amount of work but instead produces it according to some distribution with some expected work level, for example a producer of the original form:

$$Prod_z(r) = \Sigma_{i=0}^{i=r} 1.\overline{work}_z^i.\mathbf{1}.\textstyle\sum_{i \in Range} w_i.\ \mathbf{1}.Prod_z(i)$$

If we denote the expected amount of work available in task z by E_z, then when the total of these expectations is less than or equal to the total number of processes available to handle the work, we would hope that our system would allocate E_z processes to task z. Whilst this is observed in simulations it is far from easy to prove - there is no *strictly stable state.* All we can say is that if the rate at which our processes choose to change tasks is much slower than the variability in available work then the above state is stable and it will be eventually reached (but this may take a long time).

It is possible to demonstrate that if you have a cyclical work requirement and there is some point on the cycle with all the processes occupied, then provided that the cycle length is shorter than either of the values k_{lc} or k_{rc} there is a stable state with the processes in the arrangement given by relative work levels when all the processes are occupied.

5 Temporal Polyethism-Emergent Not Causal.

One of the stongest features observed about allocation of tasks within homogeneous ants is that on average the eldest ants tend to perform the more dangerous (foraging and guarding) tasks [PJ81]); younger ants working firstly on brood care followed by nest maintenance. It has been assumed that this task distribution in homogenous ants results directly from age, impling that individuals know their age and assort accordingly. Indeed it has been suggested that age stratification in ants is an "adaptive demography" where the proportion of different age groups is determined by environmental contingencies and reflects the task requirements of the species [WH88]

It has been pointed out by Calabi [Cal88] that such a mechanism would be too slow to respond to environmental variability; she presents a model that allows some short-term flexibility by age-based differential response thresholds to the various tasks. However, this model still subsumes some "knowledge" of age, and it can be argued that colonies using this mechanism would still be slow to respond to changes.

We can show that given task allocation methods outlined above then the ants performing the latter tasks will on average be older. Unfortunately we cannot do this entirely within our process calculus but will use it to justify some simplifying assumptions and to generate the understanding to produce a simulation.

We start by amending our simple task allocation system to take account of age:

$$Dead = \mathbf{1}.Dead$$

$$Task_z(k_l,k_r,a) = \begin{cases} \omega.task_{z+1}\overline{task_z}.\mathbf{1}.Task(0,0,a+1) + 1.Task'_z(k_l,k_r,a) & \text{if} \\ a < maxage & \\ Dead & \text{otherwise.} \end{cases}$$

$$\begin{aligned} Task'_z(k_l,k_r,a) = & (\omega.(1.task_{z+1}.Downdecide_z(k_l,k_r+1,a) + \\ & 1.\overline{task_z}.Updecide_z(k_l+1,k_r,a)) + \\ & 1.1.Bothdecide_z(k_l+1,k_r+1,a)) \end{aligned}$$

$$Updecide_z(k_l,k_r,a) = \begin{cases} \mathbf{1}.Task_z(k_l,k_r,a+1) & \text{if } k_r < k_{rc} \\ m.\mathbf{1}.Task_{z+1}(0,0.a+1) + n.\mathbf{1}.Task_z(k_l,k_r,a+1) & \\ \text{otherwise.} & \end{cases}$$

$$Downdecide_z(k_l,k_r,a) = \begin{cases} \mathbf{1}.Task_z(k_l,k_r,a+1) & \text{if } k_l < k_{lc} \\ m'.\mathbf{1}.Task_{z-1}(0,0,a+1) + n'.\mathbf{1}.Task_z(k_l,k_r,a+1) & \\ \text{otherwise.} & \end{cases}$$

$$Bothdecide_z(k_l,k_r,a) = \begin{cases} mn'.\mathbf{1}.Task_{z+1}(0,0,a+1) + m'n.\mathbf{1}.Task_{z-1}(0,0,a+1) + & \\ (mm'+nn').\mathbf{1}.Task_z(0,0,a+1) & \text{if } k_l \geq k_{lc} \text{ and } k_r \geq k_{rc} \\ Updecide_z(k_l,k_r,a+1) & \text{if } k_r \geq k_{rc} \text{ and } k_l < k_{lc} \\ Downdecide_z(k_l,k_r,a+1) & \text{otherwise.} \end{cases}$$

$$Task_1(k_l,k_r,a) = \omega.task_2.\mathbf{1}.Task_1(0,0,a+1) + 1.\mathbf{1}.Updecide_1(k_l,k_r,a)$$

$$Updecide_1(k_l,k_r,a) = \begin{cases} \mathbf{1}.Task_1(k_l,k_r,a+1) & \text{if } k_r < k_{rc} \\ m.\mathbf{1}.Task_2(0,0,a+1) + n.\mathbf{1}.Task_1(0,0,a+1) & \text{oth-} \\ \text{erwise.} & \end{cases}$$

$$Task_t(k_l,k_r,a) = \omega.\overline{task_t}.\mathbf{1}.Task_t(0,0,a+1) + 1.\mathbf{1}.Downdecide_t(k_l,k_r,a)$$

$$Downdecide_t(k_l,k_r,a) = \begin{cases} \mathbf{1}.Task_t(k_l,k_r,a+1) & \text{if } k_l < k_{lc} \\ m'.\mathbf{1}.Task_{t-1}(0,0,a+1) + n'.\mathbf{1}.Task_z(k_l,k_r,a+1) & \\ \text{otherwise.} & \end{cases}$$

Note that the process *Dead* is an identity for the operator $\times$. We need the following process to produce more ants:

$$Queen(n) = \underbrace{\mathbf{1}\ldots\mathbf{1}.}_{gen-times}(\prod_{i=1}^{new(n)} Task_1(0,0,0)) \times Queen(n+1)$$

Take a vector x_i of numbers in the range 1 to t then our system is the following process:

$$Nest = \Theta((\prod_{i=1}^{k} Task_{x_i}(0,0) \times Queen(1))\lceil\{\mathbf{1}\})$$

The property that we wish to establish is that the average ages of the ants performing each task ascend in the order of the tasks. Unfortunately we can see no way of proving this directly using the above description. However, we do know that even task allocations are stable some time after a new influx of young ants at $Task_0$, so if we assume that the generation time is *long* compared with the time to stability in our allocation algorithm

then we can use the following method to approximate the ages of ants performing our tasks.

For simplicity we only consider two tasks, with m and n ants performing them respectively and initially having average ages A_1 and A_2, furthermore we assume that our population (in the first task) grows in proportion k and that there is a death rate due of ageing of ϵ. We obtain the following growth equations for the average ages of our ants.

$$A_1' = \frac{\alpha n(A_1+1)(1-\epsilon A_1)}{\alpha n(1-\epsilon A_1+k)}$$
$$A_2' = \frac{m(A_2+1)(1-\epsilon A_2)+(1-\alpha)n(1-\epsilon A_1+k)A_1}{m(1-\epsilon A_2)+(1-\alpha)n(1-\epsilon A_1+k)}$$
$$\alpha = \frac{m(1-\epsilon A_2)+n(1-\epsilon A_1+k)}{(m+n)(1-\epsilon A_1+k)}$$

In the above α is the proportion in task 1 which have to move to task 2 in order to keep the number ratio balanced. If we look for a stable solution to the above we discover that

$$A_1 = \frac{1}{(k+\epsilon)}$$

and substituting $\gamma = 1 - \epsilon A_2$ and $\beta = (1 - \epsilon A_1 + k)$ we obtain the following quadratic in γ.

$$(n - (m+n))\gamma^2 + (2(m+n) - n + \epsilon n + n\beta)\gamma + n\beta - (m+n)(1+\epsilon) - n\epsilon\beta$$

This quadratic always has a root, which gives a positive solution for A_2, we take this as the stable solution for A_2 moreover the roots are not only stable, but A_2 is always greater than A_1 as required.

We include the results of a 'C' simulation of the above system (Figure 8) as it gives a better idea of scale. This simulation is for a nest with four tasks and taking each task allocation cycle (3 ticks) as 20 minutes is equivalent to 30 years of ant life (approximately 100 generations).

6 Simple Learning.

We can include a simple notion of learning by allowing ants to become more able at the tasks they perform.

$$Lrn_z(k_l, k_r, l) = p.learn_z.Tsk_z(k_l, k_r, l+1) + p'.Tsk_z(k_l, k_r, l)$$
$$Tsk_z(k_l, k_r, l) = \omega.task^l_{z+1}\overline{task_z}^l.\mathbf{1}.Lrn(0,0,l) + 1.Tsk'_z(k_l, k_r, l)$$
$$\begin{aligned} Tsk'_z(k_l, k_r, l) = &(\omega.(1.task_{z+1}.Downd_z(k_l, k_r+1, l) + \\ &1.\overline{task_z}.Upd_z(k_l+1, k_r, l)) + \\ &1.\mathbf{1}.Bothd_z(k_l+1, k_r+1, l)) \end{aligned}$$
$$Upd_z(k_l, k_r, l) = \begin{cases} \mathbf{1}.Lrn_z(k_l, k_r, l) & \text{if } k_r < k_{rc} \\ m.\mathbf{1}.Task_{z+1}(0,0,1) + n.\mathbf{1}.Lrn_z(0,0,l) & \text{otherwise.} \end{cases}$$
$$Downd_z(k_l, k_r, l) = \begin{cases} \mathbf{1}.Lrn_z(k_l, k_r, l) & \text{if } k_l < k_{lc} \\ m'.\mathbf{1}.Lrn_{z-1}(0,0,1) + n'.\mathbf{1}.Lrn_z(k_l, k_r, l) & \text{other-} \\ \text{wise.} \end{cases}$$

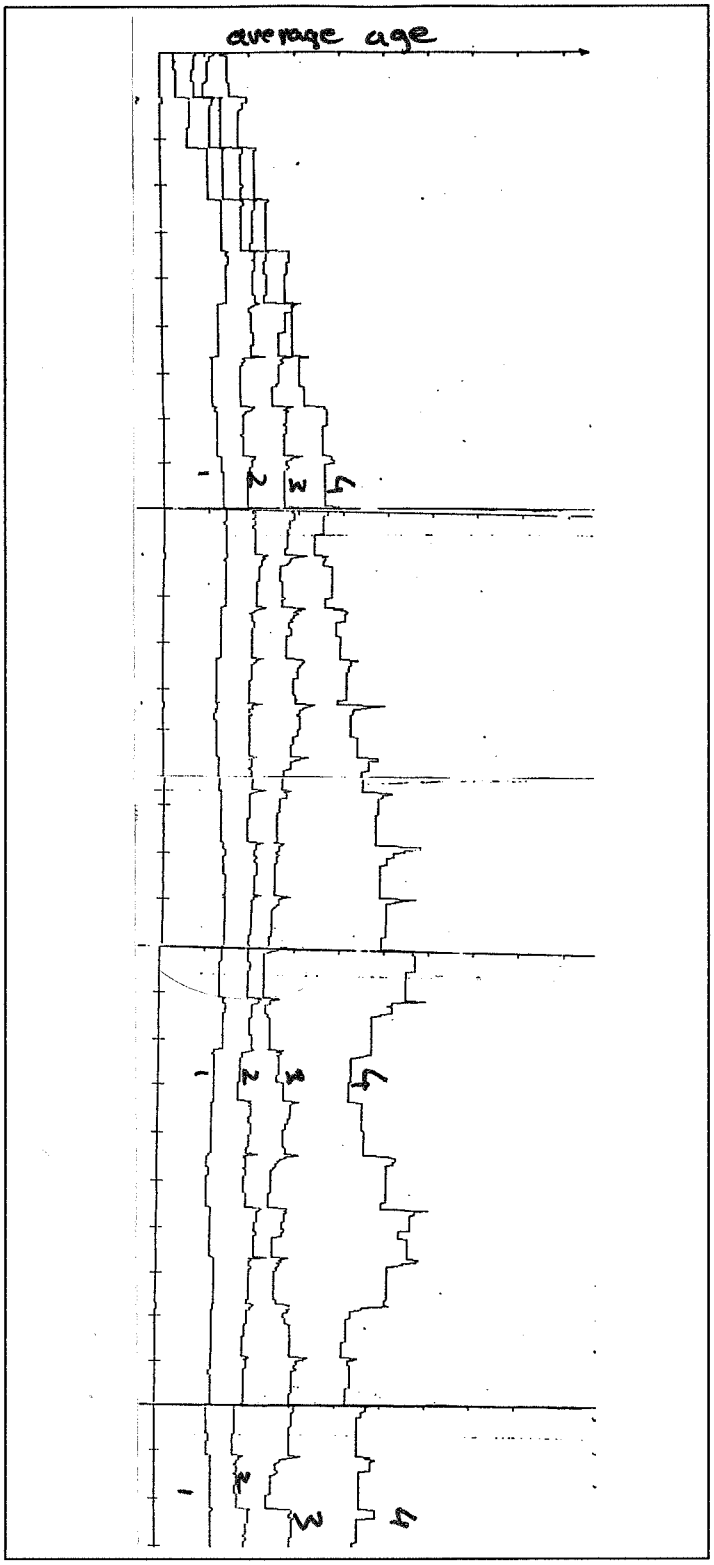

Figure 8: Average age of workers performing 4 different tasks simulated for 30 years.

$$Bothd_z(k_l,k_r,l) = \begin{cases} mn'.\mathbf{1}.Lrn_{z+1}(0,0,1) + m'n.\mathbf{1}.Lrn_{z-1}(0,0,1) + \\ (mm'+nn').\mathbf{1}.Lrn_z \text{ if } k_l \geq k_{lc} \text{ and } k_r \geq k_{rc} \\ Upd_z(k_l,k_r,l) \qquad \text{if } k_r \geq k_{rc} \text{ and } k_l < k_{lc} \\ Downd_z(k_l,k_r,l) \qquad \text{otherwise.} \end{cases}$$

$$Lrn_1(k_l,k_r,l) = p.learn_1.Tsk_1(k_l,K-r,l+1) + p'.Tsk_1(k_l,k_r,l)$$
$$Tsk_1(k_l,k_r,l) = \omega.task_2^l.\mathbf{1}.Lrn_1(0,0,l) + 1.\mathbf{1}.Upd_1$$
$$Upd_1(k_l,k_r,l) = \begin{cases} \mathbf{1}.Lrn_1(k_l,k_r,l) & \text{if } k_r < k_{rc} \\ m.\mathbf{1}.Lrn_2(0,0,1) + n.\mathbf{1}.Lrn_1(k_l,k_r,l) & \text{otherwise.} \end{cases}$$

$$Lrn_t(k_l,k_r,l) = p.learn_t.Tsk_t(k_l,k_r,l+1) + p'.Tsk_t(k_l,k_r,l)$$
$$Tsk_t(k_l,k_r,l) = \omega.\overline{task_t}^l.\mathbf{1}.Lrn_t(0,0,l) + 1.\mathbf{1}.Downd_t(k_l,k_r,l)$$
$$Downd_t(k_l,k_r,l) = \begin{cases} \mathbf{1}.Lrn_t(k_l,k_r,l) & \text{if } k_l < k_{lc} \\ m'.\mathbf{1}.Lrn_{t-1}(0,0,1) + n'.\mathbf{1}.Lrn_z(k_l,k_r,l) & \text{other-} \\ \text{wise.} \end{cases}$$

We can make learning spontaneous with probability $\frac{p}{p+p'}$ by renaming the $learn_z$ actions to $\mathbf{1}$, rendering our system thus (with the usual x_i constants):

$$Rlearn = \Theta(((\prod_{i=1}^{k} Lrn_{x_i}(0,0,1))[\mathbf{1}/learn])\lceil\{\mathbf{1}\})$$

alternatively we can include some form of teaching process which outputs $\overline{learn}$ actions and make $p = p'$ thus rendering our system as,

$$Learn = \Theta(((\prod_{i=1}^{k} Lrn_{x_i}(0,0,1))[\mathbf{1}/learn] \times Teacher)\lceil\{\mathbf{1}\})$$

In either case the stable state is achieved when the effective total of ants (that is the sum of the individual abilities of the ants in a task) is the same for all tasks, and subsequent learning is forbidden.

7 Performance with Erroneous Ants.

We examine how our system will perform in the presence of ants that make certain kinds of mistake.

7.1 Dead or Lazy Ants.

In either case the best representation for such an ant is the process *Dead* defined earlier. That is to say a process that will perform only **1** actions for ever. This is reasonable as a dead ant will not interfere with the performance of the others and a lazy ant will just sit there passing the time. Since this process is the unit for multiplication it has *no effect* on the outcome of the allocation alogrithm amongst the other ants as we would expect.

7.2 Task Fixation.

We can imagine an ant that only wishes to perform a certain task (say number z) and examine the effect that this will have on our allocation algorithm.

$$Fixed_z = \omega.task_{z+1}^{e_z}\overline{task_z}^{e_z}.\mathbf{1}.Fixed_z + \mathbf{1}.\mathbf{1}.\mathbf{1}.Fixed_z$$

So if we build our allocation system as before. Take a vector x_i of numbers in the range 1 to t then our system is the following process:

$$Fixl = \Theta((\prod_{i=1}^{k} Task_{x_i}(0,0) \times Fixed_{x_{k+1}})\lceil\{\mathbf{1}\})$$

There will be a stable state if there is a sequence of numbers $k_1, k_2, ..., k_t$ with $k_1 + \ldots + k_t = k$ and $k_1e_1 = k_2e_2 = \ldots = (k_z + 1)e_z = \ldots = k_te_t$ when our fixed process performs task z. So the presence of a fixed process merely causes the others to re-arrange themselves to re-produce a balanced arrangement.

8 Conclusions.

We have demonstrated a collection of task allocation algorithms that can be considered very accurate. They have a good tolerance to error and to the general dynamic variation within an environment. Furthermore we have found indications that the observation of age stratification in the ants performing particular tasks can result just from a simple conjunction of the fact that ants are born into one end of the production line (the brood pile), movement between tasks is ordered and that the movement is controlled randomly.

9 Acknowledgments.

Chris Tofts is supported by a B.P. Venture Research Grant. We would like to thank Melanie Hatcher, Nigel Franks, Anna Sendova-Franks, Tim Stickland, John-Louis Deneubourg and Simon Goss for their helpful discussion of the underlying algorithm. This document would have been much poorer if it had not been for interventions of David Pym and Matthew Morley. Finally I would like to thank Frantic Monkey for keeping me interested.

10 Bibliography.

[BBK86] J. Baeten, J. Bergstra and J. Klop, Syntax and defining equations for an interrupt mechanism in process algebra, Fundamenta Informatica IX, pp 127-168, 1986.

[Cal88] P. Calabi, Behavioural Flexibility in Hymenoptra: A Rexamination of the Concept of Caste, Advances in Myrmecology, E. J. Brill, Leiden.

[GSST90] R. van Glabbek, S. A. Smolka, B. Steffen and C.Tofts, Reactive, Generative and Stratified Models of Probabilistic Processes, proceedings LICS 1990.

[Kei] J. Keilson, Markov Chain Models - Rarity and exponentiality, Applied Mathematical Sciences 28, Springer Verlag.

[Hoa85] C. A. R. Hoare, Communicating Sequential Processes, Prentice-Hall 1985.

[HW90] B. Holldobler and E. O. Wilson, The Ants, Belknap Press of Harvard, University Press, 1990.

[LS89] K. G. Larsen and A. Skou. Bisimulation through probabilistic testing. proceedings POPL 1989.

[Mil83] R. Milner, Calculi for Synchrony and Asynchrony, Theoretical Computer Science 25(3), pp 267-310, 1983.

[Mil90] R. Milner, Communication and Concurrency, Prentice Hall, 1990.

[OW78] G. F. Oster and E. O. Wilson, Caste and Ecology in Social Insects, Princeton University Press, 1978.

[PJ81] S. D. Porter, C. D. Jorgensen, Foragers of the Harvester Ant, *Pogonomyrex owyheei*: a disposable Caste?, Behavioural Ecology and Sociobiology 9, pp247-256, 1981.

[Plo81] G. D. Plotkin, A structured approach to operational semantics. Technical report Daimi Fn-19, Computer Science Department, Aarhus University. 1981

[SST89] S. Smolka, B. Steffen and C. Tofts, unpublished notes. Working title, Probability + Restriction $\Rightarrow$ priority.

[THF91] C. Tofts, M.J.Hatcher, N. Franks, Autosynchronisation in Leptothorax Acervorum; Theory, Testability and Experiment, In preperation.

[Tof90a] C. Tofts, A Synchronous Calculus of Relative Frequency, CONCUR '90, Springer Verlag, LNCS 458

[Tof90b] C. Tofts, The Autosynchronisation of *Leptothorax Acervorum* (Fabricius) Described in WSCCS, LFCS-Report Number 128.

[WH88] E. O. Wilson and B. Holldobler, Dense Hetarachies and Mass Communication as the Basis of Organisation in Ant Colonies, T.R.E.E. 3 pp 65-67, 1988.