Definition (Term algebra, [23])

Let Ω be a signature and ar its type, and let X be a set (we say that the elements of X are *variables*). We define the set of *terms* over Ω with variables X, denoted by $T_\Omega(X)$, as follows:

- $X \subseteq T_\Omega(X)$, i.e., any variable is a term, and
- for any $n \in \mathbb{N}$, any $\omega \in \mathrm{ar}^{-1}[n]$, and any $t_1, t_2, \ldots, t_n \in T_\Omega(X)$, we have $\omega(t_1, t_2, \ldots, t_n) \in T_\Omega(X)$, i.e., we obtain terms by applying operations to terms.

The *term algebra* over Ω with variables X is the algebra

$$\mathcal{T}_\Omega(X) := (T_\Omega(X), (f_\omega)_{\omega\in\Omega}),$$

where

$$f_\omega : T_\Omega(X)^{\mathrm{ar}(\omega)} \to T_\Omega(X) : \left(t_1, t_2, \ldots, t_{\mathrm{ar}(\omega)}\right) \mapsto \omega\left(t_1, t_2, \ldots, t_{\mathrm{ar}(\omega)}\right).$$

Bands Named After Things From a Fictional Universe

band	bandLabel	universes	genres
wd:Q2005373	Harry and the Potters	Harry Potter universe	alternative rock, indie rock, Wizard rock
wd:Q1122398	Naglfar	Norse mythology	black metal
wd:Q2069906	Ancalagon	Tolkien's legendarium	black metal
wd:Q1888619	Azaghal	Tolkien's legendarium	black metal
wd:Q220896	Cirith Gorgor	Tolkien's legendarium	black metal
wd:Q152761	Gorgoroth	Tolkien's legendarium	black metal
wd:Q221072	Isengard	Tolkien's legendarium	black metal
wd:Q3433479	Rivendell	Tolkien's legendarium	black metal, folk metal
wd:Q11815622	Percival	The Witcher universe	contemporary folk music, pagan metal
wd:Q17399696	Methedras	Tolkien's legendarium	death metal
wd:Q221281	Morgoth	Tolkien's legendarium	death metal
wd:Q192863	Amon Amarth	Tolkien's legendarium	death metal, Viking metal
wd:Q656739	Turisas	Kalevala	folk metal
wd:Q4940449	Bombadil	Tolkien's legendarium	folk music
wd:Q758994	Audrey Horne	Twin Peaks	hard rock
wd:Q16953729	Mournblade	Elric saga	heavy metal
wd:Q220938	Cirith Ungol	Tolkien's legendarium	heavy metal
wd:Q7752593	The Mudbloods	Harry Potter universe	indie rock
wd:Q220967	Gandalf	Tolkien's legendarium	melodic death metal
wd:Q1557827	Gwen Stacy	Marvel Universe	metalcore
wd:Q398101	T'Pau	Star Trek multiverse	pop music
wd:Q479019	Isengard	Tolkien's legendarium	power metal
wd:Q678410	Marillion	Tolkien's legendarium	progressive rock, art rock, neo-progressive rock
wd:Q617627	Spock's Beard	Star Trek multiverse	progressive rock, progressive metal
wd:Q1157349	Dagoba	Star Wars expanded to other media, Star Wars universe	punk rock
wd:Q787789	Baba Yaga	Russian fairy tales	rock music
wd:Q869922	Son Goku	The World universe	rock music
wd:Q221449	Shagrat	Tolkien's legendarium	rock music
wd:Q13360300	Elbereth Taldea	Tolkien's legendarium	speed metal
wd:Q1035310	Carach Angren	Tolkien's legendarium	symphonic black metal
wd:Q3045589	Dúnedain	Tolkien's legendarium	traditional heavy metal
wd:Q49973700	Scratch Bandits Crew	Crash Bandicoot universe	turntablism, breakbeat
wd:Q57822988	CHVRN	Narnia universe	witch house
wd:Q1752517	Draco and the Malfoys	Harry Potter universe	Wizard rock
wd:Q3858512	Ministry of Magic	Harry Potter universe	Wizard rock

https://query.wikidata.org/#SELECT%20%3Fband%0A%20%20%20%20%20%20%20%3FbandLabel%0A%20%20%20%20%20%20%20%28GROUP_CONCAT%28DISTINCT%20%3FuniverseLabel%3B%20separator%3D%22%2C%20%22%29%20AS%20%3Funiverses%29%0A%20%20%20%20%20%20%20%28GROUP_CONCAT%28DISTINCT%20%3FgenreLabel%3B%20separator%3D%22%2C%20%22%29%20AS%20%3Fgenres%29%20%0AWHERE%20%7B%0A%20%20%3Fband%20wdt%3AP31%2Fwdt%3AP279%2a%20wd%3AQ215380%20%3B%0A%20%20%20%20%20%20%20%20wdt%3AP138%2F%28wdt%3AP1080%7Cwdt%3AP1434%29%2Frdfs%3Alabel%20%3FuniverseLabel%20%3B%0A%20%20%20%20%20%20%20%20wdt%3AP136%2Frdfs%3Alabel%20%3FgenreLabel%20%3B%0A%20%20%20%20%20%20%20%20rdfs%3Alabel%20%3FbandLabel%20.%0A%20%20FILTER%20%28LANG%28%3FbandLabel%29%20%3D%20%22en%22%20%26%26%20%0A%20%20%20%20%20%20%20%20%20%20LANG%28%3FgenreLabel%29%20%3D%20%22en%22%20%26%26%20%0A%20%20%20%20%20%20%20%20%20%20LANG%28%3FuniverseLabel%29%20%3D%20%22en%22%29%0A%7D%20GROUP%20BY%20%3Fband%20%3FbandLabel%20%0A%20%20ORDER%20BY%20ASC%28%3FbandLabel%29

Figure 4.3 depicts the full system.

Figure 4.3 depicts the full system. We start with one spike in the acc neuron, two spikes in the k_1 neuron, and four spikes in the kill neuron.

Figure 4.3 depicts the full system. We start with one spike in the acc neuron, two spikes in the k_1 neuron, and four spikes in the kill neuron. Initially, the first component is active, and k_1 is the only neuron where a rule can be applied.

Figure 4.3 depicts the full system. We start with one spike in the acc neuron, two spikes in the k_1 neuron, and four spikes in the kill neuron. Initially, the first component is active, and k_1 is the only neuron where a rule can be applied. One of the two rules is chosen non-deterministically, and either one or two spikes are emitted.

Figure 4.3 depicts the full system. We start with one spike in the acc neuron, two spikes in the k_1 neuron, and four spikes in the kill neuron. Initially, the first component is active, and k_1 is the only neuron where a rule can be applied. One of the two rules is chosen non-deterministically, and either one or two spikes are emitted. Then, only kill has an applicable rule.

Figure 4.3 depicts the full system. We start with one spike in the acc neuron, two spikes in the k_1 neuron, and four spikes in the kill neuron. Initially, the first component is active, and k_1 is the only neuron where a rule can be applied. One of the two rules is chosen non-deterministically, and either one or two spikes are emitted. Then, only kill has an applicable rule. For now, assume k_1 indeed sends out two spikes.

Figure 4.3 depicts the full system. We start with one spike in the acc neuron, two spikes in the k_1 neuron, and four spikes in the kill neuron. Initially, the first component is active, and k_1 is the only neuron where a rule can be applied. One of the two rules is chosen non-deterministically, and either one or two spikes are emitted. Then, only kill has an applicable rule. For now, assume k_1 indeed sends out two spikes. These two spikes are then deleted in kill, and no more rules in the first component are applicable.

Figure 4.3 depicts the full system. We start with one spike in the acc neuron, two spikes in the k_1 neuron, and four spikes in the kill neuron. Initially, the first component is active, and k_1 is the only neuron where a rule can be applied. One of the two rules is chosen non-deterministically, and either one or two spikes are emitted. Then, only kill has an applicable rule. For now, assume k_1 indeed sends out two spikes. These two spikes are then deleted in kill, and no more rules in the first component are applicable. Thus, control transfers to the second component.

Figure 4.3 depicts the full system. We start with one spike in the acc neuron, two spikes in the k_1 neuron, and four spikes in the kill neuron. Initially, the first component is active, and k_1 is the only neuron where a rule can be applied. One of the two rules is chosen non-deterministically, and either one or two spikes are emitted. Then, only kill has an applicable rule. For now, assume k_1 indeed sends out two spikes. These two spikes are then deleted in kill, and no more rules in the first component are applicable. Thus, control transfers to the second component. The neuron acc sends its contents (only a single spike for now) to d_1, d_2, and the environment, and then k_2 sends two spikes back into k_1.

Figure 4.3 depicts the full system. We start with one spike in the acc neuron, two spikes in the k_1 neuron, and four spikes in the kill neuron. Initially, the first component is active, and k_1 is the only neuron where a rule can be applied. One of the two rules is chosen non-deterministically, and either one or two spikes are emitted. Then, only kill has an applicable rule. For now, assume k_1 indeed sends out two spikes. These two spikes are then deleted in kill, and no more rules in the first component are applicable. Thus, control transfers to the second component. The neuron acc sends its contents (only a single spike for now) to d_1, d_2, and the environment, and then k_2 sends two spikes back into k_1. Then control transfers back to the first component.

Figure 4.3 depicts the full system. We start with one spike in the acc neuron, two spikes in the k_1 neuron, and four spikes in the kill neuron. Initially, the first component is active, and k_1 is the only neuron where a rule can be applied. One of the two rules is chosen non-deterministically, and either one or two spikes are emitted. Then, only kill has an applicable rule. For now, assume k_1 indeed sends out two spikes. These two spikes are then deleted in kill, and no more rules in the first component are applicable. Thus, control transfers to the second component. The neuron acc sends its contents (only a single spike for now) to d_1, d_2, and the environment, and then k_2 sends two spikes back into k_1. Then control transfers back to the first component. This process repeats until k_1 emits only a single spike.

Figure 4.3 depicts the full system. We start with one spike in the acc neuron, two spikes in the k_1 neuron, and four spikes in the kill neuron. Initially, the first component is active, and k_1 is the only neuron where a rule can be applied. One of the two rules is chosen non-deterministically, and either one or two spikes are emitted. Then, only kill has an applicable rule. For now, assume k_1 indeed sends out two spikes. These two spikes are then deleted in kill, and no more rules in the first component are applicable. Thus, control transfers to the second component. The neuron acc sends its contents (only a single spike for now) to d_1, d_2, and the environment, and then k_2 sends two spikes back into k_1. Then control transfers back to the first component. This process repeats until k_1 emits only a single spike. Assume that this happens after $n \in \mathbb{N}$ cycles of the above process.

Figure 4.3 depicts the full system. We start with one spike in the acc neuron, two spikes in the k_1 neuron, and four spikes in the kill neuron. Initially, the first component is active, and k_1 is the only neuron where a rule can be applied. One of the two rules is chosen non-deterministically, and either one or two spikes are emitted. Then, only kill has an applicable rule. For now, assume k_1 indeed sends out two spikes. These two spikes are then deleted in kill, and no more rules in the first component are applicable. Thus, control transfers to the second component. The neuron acc sends its contents (only a single spike for now) to d_1, d_2, and the environment, and then k_2 sends two spikes back into k_1. Then control transfers back to the first component. This process repeats until k_1 emits only a single spike. Assume that this happens after $n \in \mathbb{N}$ cycles of the above process. If $n > 0$, then acc has already sent

$$\sum_{i=0}^{n} 2^i = 2^n - 1$$

spikes to the environment.

Figure 4.3 depicts the full system. We start with one spike in the acc neuron, two spikes in the k_1 neuron, and four spikes in the kill neuron. Initially, the first component is active, and k_1 is the only neuron where a rule can be applied. One of the two rules is chosen non-deterministically, and either one or two spikes are emitted. Then, only kill has an applicable rule. For now, assume k_1 indeed sends out two spikes. These two spikes are then deleted in kill, and no more rules in the first component are applicable. Thus, control transfers to the second component. The neuron acc sends its contents (only a single spike for now) to d_1, d_2, and the environment, and then k_2 sends two spikes back into k_1. Then control transfers back to the first component. This process repeats until k_1 emits only a single spike. Assume that this happens after $n \in \mathbb{N}$ cycles of the above process. If $n > 0$, then acc has already sent

$$\sum_{i=0}^{n} 2^i = 2^n - 1$$

spikes to the environment. acc will now send out a final spike, bringing the total to 2^n (note that $2^0 = 1$, so this also works for $n = 0$).

Figure 4.3 depicts the full system. We start with one spike in the acc neuron, two spikes in the k_1 neuron, and four spikes in the kill neuron. Initially, the first component is active, and k_1 is the only neuron where a rule can be applied. One of the two rules is chosen non-deterministically, and either one or two spikes are emitted. Then, only kill has an applicable rule. For now, assume k_1 indeed sends out two spikes. These two spikes are then deleted in kill, and no more rules in the first component are applicable. Thus, control transfers to the second component. The neuron acc sends its contents (only a single spike for now) to d_1, d_2, and the environment, and then k_2 sends two spikes back into k_1. Then control transfers back to the first component. This process repeats until k_1 emits only a single spike. Assume that this happens after $n \in \mathbb{N}$ cycles of the above process. If $n > 0$, then acc has already sent

$$\sum_{i=0}^{n} 2^i = 2^n - 1$$

spikes to the environment. acc will now send out a final spike, bringing the total to 2^n (note that $2^0 = 1$, so this also works for $n = 0$). None of the rules in d_1 and d_2 are applicable (in fact, no rule in the system at all is applicable), so the computation stops.

Figure 4.3 depicts the full system. We start with one spike in the acc neuron, two spikes in the k_1 neuron, and four spikes in the kill neuron. Initially, the first component is active, and k_1 is the only neuron where a rule can be applied. One of the two rules is chosen non-deterministically, and either one or two spikes are emitted. Then, only kill has an applicable rule. For now, assume k_1 indeed sends out two spikes. These two spikes are then deleted in kill, and no more rules in the first component are applicable. Thus, control transfers to the second component. The neuron acc sends its contents (only a single spike for now) to d_1, d_2, and the environment, and then k_2 sends two spikes back into k_1. Then control transfers back to the first component. This process repeats until k_1 emits only a single spike. Assume that this happens after $n \in \mathbb{N}$ cycles of the above process. If $n > 0$, then acc has already sent

$$\sum_{i=0}^{n} 2^i = 2^n - 1$$

spikes to the environment. acc will now send out a final spike, bringing the total to 2^n (note that $2^0 = 1$, so this also works for $n = 0$). None of the rules in d_1 and d_2 are applicable (in fact, no rule in the system at all is applicable), so the computation stops. Clearly, the language generated by Π working according to the terminating protocol in the maximally-parallel mode and taking the total number of spikes sent to the environment is

$$C_{\mathrm{tot}}\mathcal{N}_t^{\mathrm{maxpar}}(\Pi) = \{2^n \mid n \in \mathbb{N}\} = P.$$

Figure 4.3 depicts the full system. We start with one spike in the acc neuron, two spikes in the k_1 neuron, and four spikes in the kill neuron. Initially, the first component is active, and k_1 is the only neuron where a rule can be applied. One of the two rules is chosen non-deterministically, and either one or two spikes are emitted. Then, only kill has an applicable rule. For now, assume k_1 indeed sends out two spikes. These two spikes are then deleted in kill, and no more rules in the first component are applicable. Thus, control transfers to the second component. The neuron acc sends its contents (only a single spike for now) to d_1, d_2, and the environment, and then k_2 sends two spikes back into k_1. Then control transfers back to the first component. This process repeats until k_1 emits only a single spike. Assume that this happens after $n \in \mathbb{N}$ cycles of the above process. If $n > 0$, then acc has already sent

$$\sum_{i=0}^{n} 2^i = 2^n - 1$$

spikes to the environment. acc will now send out a final spike, bringing the total to 2^n (note that $2^0 = 1$, so this also works for $n = 0$). None of the rules in d_1 and d_2 are applicable (in fact, no rule in the system at all is applicable), so the computation stops. Clearly, the language generated by Π working according to the terminating protocol in the maximally-parallel mode and taking the total number of spikes sent to the environment is

$$C_{\mathrm{tot}}\mathcal{N}_t^{\mathrm{maxpar}}(\Pi) = \{2^n \mid n \in \mathbb{N}\} = P.$$

As the only source of non-determinism in the system is the selection of the rule to be fired from among the applicable rules, it is exactly this mechanism we need to exploit to allow the system to decide which number to generate, the idea being that, depending on the rule chosen, the computation is either stopped or continues.

Figure 4.3 depicts the full system. We start with one spike in the acc neuron, two spikes in the k_1 neuron, and four spikes in the kill neuron. Initially, the first component is active, and k_1 is the only neuron where a rule can be applied. One of the two rules is chosen non-deterministically, and either one or two spikes are emitted. Then, only kill has an applicable rule. For now, assume k_1 indeed sends out two spikes. These two spikes are then deleted in kill, and no more rules in the first component are applicable. Thus, control transfers to the second component. The neuron acc sends its contents (only a single spike for now) to d_1, d_2, and the environment, and then k_2 sends two spikes back into k_1. Then control transfers back to the first component. This process repeats until k_1 emits only a single spike. Assume that this happens after $n \in \mathbb{N}$ cycles of the above process. If $n > 0$, then acc has already sent

$$\sum_{i=0}^{n} 2^i = 2^n - 1$$

spikes to the environment. acc will now send out a final spike, bringing the total to 2^n (note that $2^0 = 1$, so this also works for $n = 0$). None of the rules in d_1 and d_2 are applicable (in fact, no rule in the system at all is applicable), so the computation stops. Clearly, the language generated by Π working according to the terminating protocol in the maximally-parallel mode and taking the total number of spikes sent to the environment is

$$C_{\mathrm{tot}}\mathcal{N}_t^{\mathrm{maxpar}}(\Pi) = \{2^n \mid n \in \mathbb{N}\} = P.$$

As the only source of non-determinism in the system is the selection of the rule to be fired from among the applicable rules, it is exactly this mechanism we need to exploit to allow the system to decide which number to generate, the idea being that, depending on the rule chosen, the computation is either stopped or continues.
We depict such a module in Figure 4.2. Initially, the k_1 neuron contains two spikes and the kill neuron starts with four spikes, whereas k_2 remains empty.

Figure 4.3 depicts the full system. We start with one spike in the acc neuron, two spikes in the k_1 neuron, and four spikes in the kill neuron. Initially, the first component is active, and k_1 is the only neuron where a rule can be applied. One of the two rules is chosen non-deterministically, and either one or two spikes are emitted. Then, only kill has an applicable rule. For now, assume k_1 indeed sends out two spikes. These two spikes are then deleted in kill, and no more rules in the first component are applicable. Thus, control transfers to the second component. The neuron acc sends its contents (only a single spike for now) to d_1, d_2, and the environment, and then k_2 sends two spikes back into k_1. Then control transfers back to the first component. This process repeats until k_1 emits only a single spike. Assume that this happens after $n \in \mathbb{N}$ cycles of the above process. If $n > 0$, then acc has already sent

$$\sum_{i=0}^{n} 2^i = 2^n - 1$$

spikes to the environment. acc will now send out a final spike, bringing the total to 2^n (note that $2^0 = 1$, so this also works for $n = 0$). None of the rules in d_1 and d_2 are applicable (in fact, no rule in the system at all is applicable), so the computation stops. Clearly, the language generated by Π working according to the terminating protocol in the maximally-parallel mode and taking the total number of spikes sent to the environment is

$$C_{\mathrm{tot}}\mathcal{N}_t^{\mathrm{maxpar}}(\Pi) = \{2^n \mid n \in \mathbb{N}\} = P.$$

As the only source of non-determinism in the system is the selection of the rule to be fired from among the applicable rules, it is exactly this mechanism we need to exploit to allow the system to decide which number to generate, the idea being that, depending on the rule chosen, the computation is either stopped or continues.
We depict such a module in Figure 4.2. Initially, the k_1 neuron contains two spikes and the kill neuron starts with four spikes, whereas k_2 remains empty. If k_1 emits two spikes, these two are then forgotten in kill.

Figure 4.3 depicts the full system. We start with one spike in the acc neuron, two spikes in the k_1 neuron, and four spikes in the kill neuron. Initially, the first component is active, and k_1 is the only neuron where a rule can be applied. One of the two rules is chosen non-deterministically, and either one or two spikes are emitted. Then, only kill has an applicable rule. For now, assume k_1 indeed sends out two spikes. These two spikes are then deleted in kill, and no more rules in the first component are applicable. Thus, control transfers to the second component. The neuron acc sends its contents (only a single spike for now) to d_1, d_2, and the environment, and then k_2 sends two spikes back into k_1. Then control transfers back to the first component. This process repeats until k_1 emits only a single spike. Assume that this happens after $n \in \mathbb{N}$ cycles of the above process. If $n > 0$, then acc has already sent

$$\sum_{i=0}^{n} 2^i = 2^n - 1$$

spikes to the environment. acc will now send out a final spike, bringing the total to 2^n (note that $2^0 = 1$, so this also works for $n = 0$). None of the rules in d_1 and d_2 are applicable (in fact, no rule in the system at all is applicable), so the computation stops. Clearly, the language generated by Π working according to the terminating protocol in the maximally-parallel mode and taking the total number of spikes sent to the environment is

$$C_{\mathrm{tot}}\mathcal{N}_t^{\mathrm{maxpar}}(\Pi) = \{2^n \mid n \in \mathbb{N}\} = P.$$

As the only source of non-determinism in the system is the selection of the rule to be fired from among the applicable rules, it is exactly this mechanism we need to exploit to allow the system to decide which number to generate, the idea being that, depending on the rule chosen, the computation is either stopped or continues.

We depict such a module in Figure 4.2. Initially, the k_1 neuron contains two spikes and the kill neuron starts with four spikes, whereas k_2 remains empty. If k_1 emits two spikes, these two are then forgotten in kill. Control transfers to the second component, k_2 emits the two spikes, control transfers back to the first component, and the module has returned to its initial configuration.

Figure 4.3 depicts the full system. We start with one spike in the acc neuron, two spikes in the k_1 neuron, and four spikes in the kill neuron. Initially, the first component is active, and k_1 is the only neuron where a rule can be applied. One of the two rules is chosen non-deterministically, and either one or two spikes are emitted. Then, only kill has an applicable rule. For now, assume k_1 indeed sends out two spikes. These two spikes are then deleted in kill, and no more rules in the first component are applicable. Thus, control transfers to the second component. The neuron acc sends its contents (only a single spike for now) to d_1, d_2, and the environment, and then k_2 sends two spikes back into k_1. Then control transfers back to the first component. This process repeats until k_1 emits only a single spike. Assume that this happens after $n \in \mathbb{N}$ cycles of the above process. If $n > 0$, then acc has already sent

$$\sum_{i=0}^{n} 2^i = 2^n - 1$$

spikes to the environment. acc will now send out a final spike, bringing the total to 2^n (note that $2^0 = 1$, so this also works for $n = 0$). None of the rules in d_1 and d_2 are applicable (in fact, no rule in the system at all is applicable), so the computation stops. Clearly, the language generated by Π working according to the terminating protocol in the maximally-parallel mode and taking the total number of spikes sent to the environment is

$$C_{\mathrm{tot}}\mathcal{N}_t^{\mathrm{maxpar}}(\Pi) = \{2^n \mid n \in \mathbb{N}\} = P.$$

As the only source of non-determinism in the system is the selection of the rule to be fired from among the applicable rules, it is exactly this mechanism we need to exploit to allow the system to decide which number to generate, the idea being that, depending on the rule chosen, the computation is either stopped or continues.

We depict such a module in Figure 4.2. Initially, the k_1 neuron contains two spikes and the kill neuron starts with four spikes, whereas k_2 remains empty. If k_1 emits two spikes, these two are then forgotten in kill. Control transfers to the second component, k_2 emits the two spikes, control transfers back to the first component, and the module has returned to its initial configuration. If k_1 emits only a single spike, however, kill emits five spikes, and computation then stops as none of the neurons contains an applicable rule anymore.

Figure 4.3 depicts the full system. We start with one spike in the acc neuron, two spikes in the k_1 neuron, and four spikes in the kill neuron. Initially, the first component is active, and k_1 is the only neuron where a rule can be applied. One of the two rules is chosen non-deterministically, and either one or two spikes are emitted. Then, only kill has an applicable rule. For now, assume k_1 indeed sends out two spikes. These two spikes are then deleted in kill, and no more rules in the first component are applicable. Thus, control transfers to the second component. The neuron acc sends its contents (only a single spike for now) to d_1, d_2, and the environment, and then k_2 sends two spikes back into k_1. Then control transfers back to the first component. This process repeats until k_1 emits only a single spike. Assume that this happens after $n \in \mathbb{N}$ cycles of the above process. If $n > 0$, then acc has already sent

$$\sum_{i=0}^{n} 2^i = 2^n - 1$$

spikes to the environment. acc will now send out a final spike, bringing the total to 2^n (note that $2^0 = 1$, so this also works for $n = 0$). None of the rules in d_1 and d_2 are applicable (in fact, no rule in the system at all is applicable), so the computation stops. Clearly, the language generated by Π working according to the terminating protocol in the maximally-parallel mode and taking the total number of spikes sent to the environment is

$$C_{\mathrm{tot}}\mathcal{N}_t^{\mathrm{maxpar}}(\Pi) = \{2^n \mid n \in \mathbb{N}\} = P.$$

As the only source of non-determinism in the system is the selection of the rule to be fired from among the applicable rules, it is exactly this mechanism we need to exploit to allow the system to decide which number to generate, the idea being that, depending on the rule chosen, the computation is either stopped or continues.

We depict such a module in Figure 4.2. Initially, the k_1 neuron contains two spikes and the kill neuron starts with four spikes, whereas k_2 remains empty. If k_1 emits two spikes, these two are then forgotten in kill. Control transfers to the second component, k_2 emits the two spikes, control transfers back to the first component, and the module has returned to its initial configuration. If k_1 emits only a single spike, however, kill emits five spikes, and computation then stops as none of the neurons contains an applicable rule anymore.

4.4. Comparing the two approaches

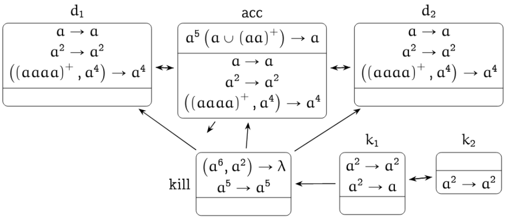

Figure 4.3.: The complete system Π

Expansion rules

Use	Expansion
`Ts...`	`T1, ..., Tn`
`Ts&&...`	`T1&&, ..., Tn&&`
`x<Ts, Y>::z...`	`x<T1, Y>::z, ..., x<Tn, Y>::z`
`x<Ts&, Us>...`	`x<T1&, U1>, ..., x<Tn&, Un>`
`func(5, vs)...`	`func(5, v1), ..., func(5, vn)`

- (Please note: ellipses on the right are in a different font)

Abstract

- It was a simple choice, really, on an IBM 370 in the 70's, between APL, Fortran, Lisp 1.5, PL/1, COBOL, and Simula'67. Nothing could come close to Simula's combination of strong typing, garbage collection, and proper string processing. Separate compilation (prefix classes) and coroutines were nice bonuses. And then there were these ... "objects" but, well, nothing is perfect. Hot topics in those days were the freshly invented denotational semantics (which Simula didn't have), formal type systems (which objects didn't have), and abstract data types (which seemed to have confusingly little to do with classes). Still, Simula was the obvious choice to get something done comfortably because, after all, it was an improved Algol. It even supported the functional programming feature of call-by-name. So, it became my first favorite language, for every reason other than it being object-oriented.
- The story I am going to tell is the very, very slow realization that Simula was the embodiment of a radically different philosophy of programming, and the gradual and difficult efforts to reconcile that philosophy with the formal methods that were being developed for procedural and functional programming. Along the way, domain theory helped rather unexpectedly, at least for a while. Type theory had to be recast for the task at hand. Landin's lambda-reductionism had to be partially abandoned. Always, there seemed to be a deep fundamental mismatch between objects and procedures, well described by Reynolds, that made any unification impossibly complicated. But in the end, both object-oriented and procedural programming have benefited from the clash of cultures. And the story is far from over yet, as witnessed by the still blooming area of program verification for both procedural and object-oriented languages.