A Process Algebraic Approach to Computational Linguistics

Tsutomu Fujinami

A thesis submitted in fulfilment of the requirements
for the degree of Doctor of Philosophy
to the
University of Edinburgh
1996

Abstract

The thesis presents a way to apply *process algebra* to computational linguistics.[1] We are interested in how *contexts* can affect or contribute to language understanding and model the phenomena as a system of communicating processes to study the interaction between them in detail. For this purpose, we turn to the π-calculus and investigate how communicating processes may be defined. While investigating the computational ground of communication and concurrency, we devise a graphical representation for processes to capture the structure of interaction between them. Then, we develop a logic, combinatory intuitionistic linear logic with equality relation, to specify communicating processes logically. The development enables us to study Situation Semantics with process algebra. We construct semantic objects employed in Situation Semantics in the π-calculus and then represent them in the logic. Through the construction, we also relate Situation Semantics with the research on the information flow, Channel Theory, by conceiving of linear logic as a theory of the information flow. To show how sentences can be parsed as the result of interactions between processes, we present a concurrent chart parser encoded in the π-calculus. We also explain how a semantic representation can be generated as a process by the parser. We conclude the thesis by comparing the framework with other approaches.

[1]The ideas presented in the dissertation have been presented at three occasions. The idea to apply process algebra to studying Situation Theory was first presented at the first International Workshop on Computational Semantics held at the Institute for Language Technology and Artificial Intelligence, Tilburg University, The Netherlands in December 1994 [Fuj94]. The idea was refined and presented also at the 10th Amsterdam Colloquium held at ILLC/Department of Philosophy, University of Amsterdam, The Netherlands in December 1995 [Fuj96b]. Another idea to apply process algebra to studying feature structures and parsing was presented at the second Conference on Information-Theoretic Approaches to Logic, Language, and Computation held at London Guildhall University in July 1996 [Fuj96a].

Declaration

This thesis has been composed by myself and it has not been submitted in any previous application for a degree. The work reported within was executed by myself, unless otherwise stated.

August 1996

Acknowledgements

The ideas exploited in the thesis have been developed through a number of discussions with people who have influenced me in various ways. I owe most to Robin Cooper as my first supervisor, with whom I often discussed how process algebra could be applied to studying Situation Semantics. Jonathan Ginzburg as my second supervisor helped me notice there was an interesting problem in the use of the Japanese particle, *tte*, which motivated me to look into process algebra. The series of meetings with Robin and Jonathan, where David Milward was also present, were inspiring and directed me to investigating the problem further. Yukinori Takubo, who had already investigated the use of the particle, kindly sent me his papers with valuable comments. Megumi Kameyama also helped me notice there was a dynamic aspect in the use of the particle. Jean Carletta pointed out the particle had to do with repair and motivated me to study the phenomenon in broader contexts. My interest towards concurrency can be traced back to the time when I worked for a company. Toshinori Watanabe used to encourage me to look into concurrency and convinced me that it was the most important area of study. I was lucky for an encounter with Robin Milner, with whom I met at a party following a concert. Although we did not talk about concurrency at all, his enthusiasm to Scarlatti and harpshicords was impressive enough to tempt me to buy his book. Benjamin Pierce and David Turner were helpful while I was programming a parser in PICT language. The discussion with Masahito Hasegawa was useful to sort out my ideas on Linear Logic, the π-calculus, and Category Theory. While I was thinking of applying my ideas to parsing, Gert Smolka kindly taught me about their parser encoded in Oz language. Esther König-Baumer advised me to encode a chart parser. I am also thankful to Suresh Manandhar for his advice. The conversation with Alan Black was inspiring and affected me in shaping up my research program. The Map Task corpora were valuable sources to check my idea against real conversations. I thank people who have worked for these corpora, people at HCRC and Chiba University. As for the Japanese version, I am specially thankful to Syun Tutiya and Hanae Koiso, who provided me with the corpus and related documents. Masato Ishizaki helped me look into these corpora with valuable advice. I finally thank people who helped me come to Edinburgh and go through a difficult time. The fun and joy I have got from the series of concerts at St. Cecilia's Hall made it easy to adapt myself to unfamiliar circumstances and

notorious Scottish weather. I thank John and Sheila Barnes, who introduced me to Georgian Concert Society and invited me sometimes to their home for a musical experience. I owe much to my wife, Yoshiko. Without her help, I could not finish my study at all. I am thankful to my father and mother in law, who understood my purpose for study and supported our life in Edinburgh. I am also thankful to my mother and late father. I hope they will be amazed to know their genetic code played some role while the thesis was materialising.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis presents a formal theory of language use as actions with the emphasis on the notions of *communication* and *concurrency*. Our theory is based on process algebra, namely the π-calculus [Mil93b, MPW92], with which we model the process of language understanding, i.e., parsing, interpreting, and evaluating utterances. We start this introduction by clarifying our approach, then discuss how our theory may contribute to understanding language use.

1.1 Language use as actions

We investigate actual utterances rather than our capability of understanding sentences. The approach is called *performance* theory and is contrasted with *competence* theory advocated by Noam Chomsky [Cho65], where our capability of understanding sentences is the target of study, not actual utterances. Although the latter approach has been dominant in the development of linguistic theories, some people, especially people working on dialogue [Cla92], are turning to the former approach, looking into actual utterances to study language use. We follow this move and propose a way to study linguistic performance, not of competence.

To investigate actual utterances will raise interesting issues to our study, which was not paid much attention in competence theory. When we look into a corpus recording actual conversations, for example, we can soon recognise that people do not always utter complete sentences. They often pose their utterance, waiting for a response from the other, or even they may be interrupted by the other before completing a sentence. They may stop uttering a sentence in the middle and restart uttering a different sentence. Sentences are often not well-formed and may not even be a sentence, yet people can proceed with their conversation to understand each other. This sort of phenomena go beyond the scope of competence theory, but how can we study them, then?

Our strategy is to view language use as a class of human actions to communicate with each other as proposed by *speech acts* theorists such as John Austin [Aus62] and to study it from broader perspectives, where utterances are considered as just one way of conveying information from one to another. The key is that people can get themselves understood with various means, not limited to language, e.g., pointing an object. The strategy forces us, on one hand, to set up a general theory of information flow and requires us, on the other hand, to test the theory against real data. In this thesis, we confine ourselves to building the theory and will not test it against data. We explain, however, our motivation of the project through the examination of linguistic data, which we believe can be modelled with our approach.

1.2 Contexts: Linguistic data

The factors involved in actual utterances but not in sentences in themselves are often termed as '*contexts*'. Contexts are thought to determine or at least influence the meaning and content of sentences uttered in a particular occasion. As the term is used in so many different ways, we specify it into just three factors: *circumstances*, *plans and goals*, and *discourse*. These are explained in the following through the examination of excerpts from the HCRC (the Human Communication Research Centre) Map Task Corpus [ABB+91] and the Japanese Map Task Corpus [AIK+94]. We point out for each factor that *communication* and *concurrency* play a central role.

The Map Task is designed at HCRC, University of Edinburgh, to record spontaneous dialogue between English native speakers. In the task, the route giver (G) and the route follower (F) are given slightly different maps. For example, an item may be missing in one's map. The same item may be labelled with different names. They are then required to draw the correct route on the follower's map only through conversation and eye contact.[1] The giver knows their starting point, goal, and the route, while the follower knows only the starting point. The differences between their maps may make the task difficult. Some items are missing on the other's map, and other items may be labelled differently. The Japanese Map Task Corpus is designed and recorded at Chiba University, Japan, following the HCRC project in principle but for a different target language, Japanese.

1.2.1 Circumstances

One difference between utterances and mere sentences is that uttering a sentence involves the circumstance that the sentence concerns, which may influence the meaning and content. The

[1] They may not be allowed for eye contact in a different setting.

same sentence and word may mean different things under different circumstances. Without identifying the circumstance, we cannot understand what a sentence means in reality, its *content*. In the Map Task, the circumstances are fixed by isolating dialogue participants into a room, and their maps comprise most of their circumstances. Given maps as their circumstances for dialogue, one has to relate labels uttered by another with actual items in his map. Thus, maps form part of contexts for their conversation.

As for circumstances, we are interested in the way they are utilised by participants during conversations.[2] An observation reveals that participants can extract information in parallel from circumstances while they are listening or talking to others. In the excerpt shown in the figure 1.1, the giver is conducting the follower to take the correct route along the west lake, which both can see on their maps. It seems that the follower, hearing the giver's utterance, *"a half shape"* (G_3), is trying to pencil in the route on his map while he is uttering F_3 and F_4. Replying to the follower, the giver must be paying his attention to the route on his map while he is listening to the follower's utterances and replying to him (G_4 and G_5).

The same thing can happen regardless of the language they use. Figure 1.2 is an excerpt from the Japanese Map Task Corpus in the same situation. While they are discussing the shape of the lake (from G_3 to F_4), they are clearly paying their attention to the lake on their maps. These evidences should be enough to understand that circumstances serve as contexts while participants are engaged in a conversation and that they can extract information from their circumstances in parallel to their talking or listening to others.

1.2.2 Plans and goals

Plans and goals that participants have in their mind can influence their utterances and understandings. Observe another excerpt from the HCRC Map Task Corpus (Figure 1.3) with different maps from the above, where the giver is trying to conduct the follower to come to the level of a lake. In G_3, the giver first tries to conduct the follower directly to come to the level, but realizes that he may not recognise the lake and interrupts his utterance (*"walk right up by ..."*). He changes now his plan[3] in G_4 and asks if the follower has the lake in his map by saying *"do you have a lake?"* After confirming that the follower has it, the giver resumes his plan in G_5. This is an example where plans can influence utterances.

Plans and goals can also influence their understandings. In another excerpt from the Japanese Map Task Corpus (Figure 1.4), which is another part of the session shown in Figure 1.2, the

[2] Another important issue is how an utterance is systematically evaluated under different circumstances, but it is not the aim of this thesis.

[3] More accurately, this is a plan for *discourse*, which should be distinguished from that for *tasks*. I owe to Syun Tutiya in noticing the distinction.

G_1: Okay. We're going straight due north. At the top there there's a west lake

F_1: Okay.

G_2: which we're going to pass on the south *erm* southeast

F_2: *Mmhmm.*

G_3: side. And we're gonna do that in a curve, almost a half shape.

F_3: A half shape

G_4: Yeah.

F_4: to the southeast?

G_5: Yeah

F_5: Right, okay.

G_6: the southeast. And continue up north slightly.

Figure 1.1. An excerpt from the HCRC Map Task Corpus (1)

G_1: `toriaezu migi-hashi-no`
try righthand side

F_1: `migi-hasi`
righthand side

G_2: `hai, mitizoi-ni zutto itte-eh`
yes, along the route we are going straight for a while

F_2: `itte-eh`
We are going straight

G_3: `de, minami-no-numa-no-katati-ga nan-ka`
then, the form of the west lake looks... as if

F_3: `nan-darou nan-ka kou ue-no hou-de 2-tu-no kou`
well as if two things upwards are...

G_4: `wakarete-masu-yo-ne-eh`
branching, isn't it?

F_4: `eh eh eh`
yeah, yeah

Figure 1.2. An excerpt from the Japanese Map Task Corpus (1)

G$_1$: Have you got a white mountain?

F$_1$: Yes, I have a white

G$_2$: Right. Up

F$_2$: mountain to my north.

G$_3$: past ... Up on the left of the white mountain, walk right up by ...

F$_3$: *Uh-huh*

G$_4$: *Erm*, do you have a lake? Right up at the top?

F$_4$: Yes, *uh-huh.*

G$_5$: Right. Walk ... Keep walking north until you get to the level of the lake.

F$_5$: *Uh-huh.*

Figure 1.3. An excerpt from the HCRC Map Task Corpus (2)

giver tries to take the follower to near a lake. The giver at G$_3$ uses the word `mann-naka` (centre[4]) to express the point to which the follower should come to, the point that is shown as **A** in Figure 1.5. The point is in centre or middle relative to the side it faces. The meaning of the word is however ambiguous as it can mean another point **B**, too, in Japanese, the centre inside the lake. Realising the possibility of misunderstanding, the giver is looking for another appropriate word. The follower however can disambiguate the meaning in F$_3$. The disambiguation is easy as the route cannot go into the lake in the task. The giver's joke and the follower's reaction in G$_4$ and F$_4$, respectively, indicate that they are certainly aware of the ambiguity and that they could disambiguate it. We see in the example that the plan[5] for tasks may contribute to disambiguating expressions.

1.2.3 Discourse

For the moment, we use the term, *discourse*, in a very narrow sense, to mean all sentences uttered up to an utterance. Discourse provides for references to succeeding utterances. Observe the other excerpt from the HCRC Map Task Corpus (Figure 1.6). The demonstrative, *"that"*, does not make sense without its preceding sentence, which refers to the *"abandoned cottage"*.

[4] In English, it could be better translated to *middle*, so that the two points can be distinguished lexically. But the word can mean both points in Japanese.

[5] This sort of plan should be distinguished from discourse plan, whose example we have seen in English in this subsection.

F$_1$: `jaa, golf-jo-no migigawa-wo` `toht-te`
then, righthand side of the golf course *we are going*

G$_2$: `migigawa-wo toht-te-tte` `kita-no numa-no`
we are going righthand side, and *as for the east lake*

F$_2$: `kita-no numa-no`
as for the east lake

G$_3$: `nan-ka, mann-naka` `nan-tuun-desu-ka` `kita-no numa-made tuitara`
well, centre *how can I say,* *when you come to the east lake*

F$_3$: `un, mann-naka` `kita-no numa-no mann-naka-ni` `toriaezu ike-ba iin-desu-ne`
yeah, centre *to the centre of the east lake* *I should go, shouldn't I?*

G$_4$: `numa-ni` `okkoccha dame-desu-yo (laugh)`
into the lake *you shouldn't go (laugh)*

F$_4$: `(laugh) az, hai, ikimashi-ta.`
(laugh) yes, I've come to.

Figure 1.4. An excerpt from the Japanese Map Task Corpus (2)

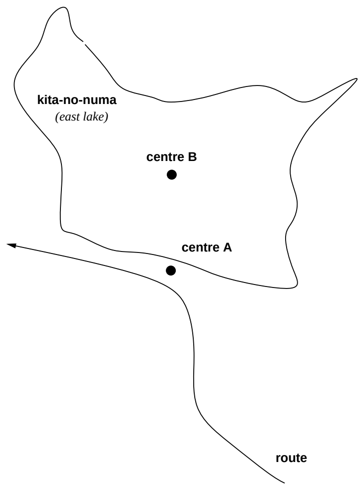


Figure 1.5. `kita-no-numa`(east lake) in the map

Another interesting case can be found in Figure 1.7. This time, the follower's utterance, F_2, *"Over the top of the?"*, does not make sense in itself. Unless the discourse is given, it cannot be understood as referring to part of the preceding sentence. The next utterance of the giver, *"slate mountain?"* (G_3), too, needs the discourse to make sense.

G_1: And immediately below that bend there is an abandoned cottage.

F_1: Okay

G_2: And we're passing above the top of <u>that</u>. We're going to continue in that sort of shape, a big wide.

F_2: *Mmhmm*

Figure 1.6. An excerpt from the HCRC Map Task Corpus (3)

G_1: When you get to the savannah, before you turn to your left, so you're heading north again.

F_1: Okay.

G_2: Until you get over the top of the slate mountain.

F_2: Over the top of the?

G_3: slate mountain?

F_3: Don't have a slate mountain.

G_4: *Mmhmm*

Figure 1.7. An excerpt from the HCRC Map Task Corpus (4)

Discourse plays an important role in Japanese, too. In the other excerpt from the Japanese Map Task Corpus (Figure 1.8), one can observe that at F_1 the follower repeats part of the sentence prior to it, `hidari-naname-mae`, as is observed in English.

1.3 The information flow by utterances

In the above, we have seen that *contexts*, i.e., circumstances, plans and goals, and discourse, can influence the way that the meaning and content of utterances are understood. In this section, we fit them into a general picture of information flow by utterances.

G_1: `ee-to, 90-do-made ika-nai-n-desu-kedo` `chotto hidari-yori-ni`
eh, not as wide as right angle, but *a little lefthand side*
`hidari-naname-mae-gurai-ni magaru-n-desu-ne`
we are going to turn forward lefthand side.

F_1: `hidari-naname-mae?`
forward lefthand side?

G_2: `ee, sinkou-houkou`
yes, in forward direction

Figure 1.8. An excerpt from the Japanese Map Task Corpus (3)

1.3.1 Linguistic channels

The figure 1.9 depicts our view on the information flow by utterances, i.e., how information can be conveyed from one to another by utterances. In our view, a *particular* utterance denotes or signals a *particular* situation, the situation that the utterance intends to convey to another. We call the relation between an utterance and a situation *signalling* relation. To recognise the signalling relation, participants must classify it as a particular belonging to a certain type.

To classify a particular signalling relation between a particular utterance and a situation, one has first to categorise the utterance into a sentence type. We call the phase as *parsing*. Sentence types can be expressed, for example, with syntactic structures and lexical information of each word. Syntactic theories are concerned with the sentence type and parsing phase. Given a sentence type, one can relate it with a situation type, whose phase we call *interpretation*. We also call the phase of relating a situation type with a particular situation as *evaluation*.

We owe Situation Theory [BP83] and Channel Theory [Bar93, BS92, SB93] for depicting the above picture. The emphasis on the distinction between particulars and types are central to Situation Theory. We inherit the idea in distinguishing utterances from sentence types and situations from situation types. The relation between sentence types and situation types are often termed as *constraints* in the theory, whose concept is further developed in Channel Theory. Constraints can also be called *indicating* relation. The distinction between signalling and indicating relation is central to Channel Theory. They form jointly *channels*. Respecting the terminology of the theory, we will call the steps depicted in the figure *linguistic channels*.

1.3.2 Contexts in linguistic channels

Into the linguistic channels, the three factors that we have listed as comprising contexts can be fitted as shown in Figure 1.10. Discourse serves as a context to parsing phase, plans and

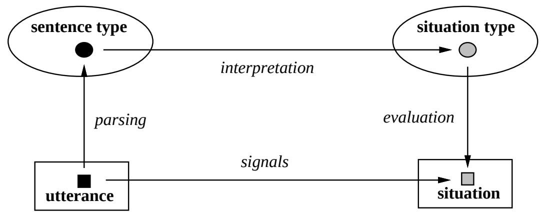


Figure 1.9. The information flow by utterances

goals to interpretation phase, and circumstances to evaluation phase. In the previous section, we have observed that circumstances can determine the content of utterances, that plans and goals can influence the meaning, i.e., situation type, and that discourse can play a role in identifying sentence types of utterances. There could also be interactions between these three factors of discourse, plan-goals, and circumstances, through which more complex forms of interactions between the channel and circumstances may occur. We will however simplify our conceptualisation in the thesis.

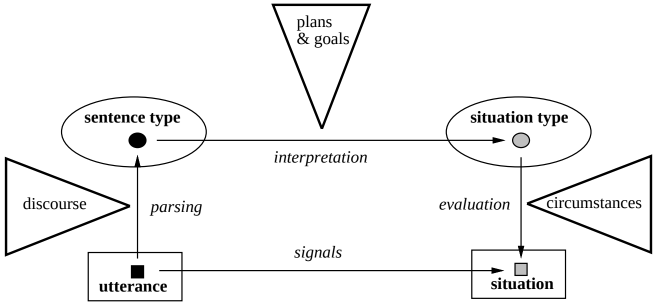


Figure 1.10. Contexts in the information flow

Given the above picture, we can ask three important questions:

1. how can discourse interact with the parsing phase?
2. how can plans and goals interact with the interpretation phase?, and
3. how can circumstances interact with the evaluation phase? enumerate To answer to

these questions, however, we have to investigate how the following items of information are organised and represented within an agent:

- sentence types, situation types, and
- discourse, plans and goals, and circumstances.

These issues should be studied with consideration to the above three questions. That is,

(a) Sentence and situation types must be organised and represented in the way that they can interact with discourse, plans and goals, and circumstances at any point, and

(b) Discourse, plans and goals, and circumstances, must be represented in the way that the phases of parsing, interpreting, and evaluating utterances can interact with them at any point.

1.3.3 Communication and Concurrency

The three questions and the two desiderata above lead us to the notions central to our theory, *communication* and *concurrency*. In our approach, we will conceive of linguistic channels as a system of communicating processes, where various objects such as sentence types, situation types, circumstances, and situations are modelled as processes, too. The systems can be conventionally regarded as consisting of three subsystems to parse, interpret, and evaluate utterances, respectively:

- The first subsystem parses utterances and is composed of four sort of processes: the first one for representing utterances, the second one for parsing utterances, third one for representing discourse, and the fourth for representing sentence types.
- The second subsystem interprets sentence types and is composed of four sort of processes. They represent sentence types, plans and goals, and situation types, and organise processes representing situation types. The processes representing sentence types are shared with the first subsystem.
- The third subsystem evaluates situation types and is composed of four sort of processes: the first one for representing situation types, the second for representing circumstances, the third for organising processes representing situation types, and the fourth for representing situations. The first sort of processes representing situation types are shared with the second subsystem.

1.4 Repair and operational semantics of natural language

While our modelling of language use enables us to study the spontaneous information flow brought about by utterances and the influence of contexts on it in detail, a question remains to be asked: How can the modelling contribute to the study of natural language semantics? Put the question differently, what aspects of language use can be captured better in the long run with our approach? In the thesis, we confine ourselves to studying *how* information can be conveyed from one to another by utterances rather than investigating what the meaning of a particular class of sentences is. The latter sort of study consists of the mainstream of research in natural language semantics. Given the tendency, one may ask why we have to study the information flow by utterances, wondering whether such an enterprise is necessary for our understanding of language use.

It may not be problematic how information can be conveyed as long as we are only concerned with language use in ideal environments. Every utterance can convey to another what it is intended to mean without failure. This is however not always the case in our real life; One may not hear the other properly, may not understand what he means, or may not relate the meaning to particular situations. We think our approach makes it possible to study the meaning of sentences when there is a failure in communication.

We are particularly interested in *repair* of common ground in dialogue in this respect. The notion of repair can be for example explained by Schegloff as

> Past work has given strong indications of a fundamental form of organization in talk-in-interaction that provides mechanisms for the participants to deal with an immense variety of troubles in speaking, hearing, or understanding the talk. These range from inability to access a word when needed or to articulate it property, to transient problems in hearing (e.g., due to ambient noise), to variously based problems of understanding; the *"variety of troubles"* thus includes various classes of problems and a virtually unlimited array of *"sources"* or *"causes."* This *"self-righting mechanism"* that allows talk-in-interaction to keep itself going in the face of such *"problems"* we have termed the organization of *repair*. A brief resumé of some main features of the organisation of repair will provide the context for the specific concerns of this article.([Sch92]: Appendix A)

The challenging question in terms of semantics is what the meaning of sentences is when they are uttered for repairing their common ground. They cannot denote any extensional entities as participants may be discussing what they are. They cannot be intensional as participants may negotiate what they mean. Extensions and intensions must be somehow fixed so that they could be used as reference points, but they are in flux while participants are repairing common ground. The term, *floating arena*, named by Schegloff [Sch92], expresses the nuance of the problem well. The notion of *awareness/vividness* studied by Carletta [Car92], too, addresses the issue from a different perspective. We expect we can approach the problem from operational point of view. Before proceeding to discussion, we examine examples to clarify the issue.

1.4.1 A Japanese phrase-final particle, *"tte"*

We examine the use of a Japanese phrase-final particle, *"tte"*, because it is an example where repair is evident in linguistic expression. In the excerpt from the Japanese Map Task Corpus shown in Figure 1.11, the participants have the same item but named differently: `rotenburo` (*hot wells*) in the giver's map and `iwaburo` (*hot springs*) in the follower's. Just before the conversation starts, they have found that the follower has not a mountain in his map, while the giver has it in his map, and are consequently alerted. To proceed with their conversation, the giver asks G_1 if the follower has `rotenburo` (*hot wells*) in his map, marking the noun phrase with the particle, *"tte"*. His expectation is confirmed by his response, F_1, and the fact becomes commonly known that the giver has `rotenburo` (*hot wells*) while the follower does not. Looking at his map, the follower informs him that he has instead `iwaburo` (*hot springs*), marking the phrase with the particle.

G₁: `jaa, rotenburo-tte arimasu?`
then, do you have hot wells?(`meta`)

F₁: `rotenburo-mo nai-mitai`
It seems that I don't have hot wells either.

G₂: `naiz,`
eh, you don't have it.

F₂: `demo, iwaburo-tte-iu ofuro-wa aru`
but I have wells called hot springs.(`meta`)

G₃: `iwaburo, ettoh, sorekanah`
hot spring! ah, that may be it.

Figure 1.11. An example of the use of *"tte"* (1)

One explanation has been proposed by Takubo [Tak89, Tak90, TK92] on the meaning of the particle, who observed it can mark noun phrases whose referent or meaning is not known to a participant engaged in the dialogue whoever he is, the speaker or the hearer. As the meaning concerns in a sense the *mode* of meaning, shared or unshared, he called noun phrases marked with the particle *meta-linguistic* expression, whose terminology we follow here. What makes it difficult to understand the meaning of the particle is that the speaker can mark a noun phrase with it even when he knows its meaning and referent if he thinks the hearer does not, as is observed in the above excerpt (Figure 1.11). The factor can be explained as an example of collaboration between participants or emphaty to the hearer [Kun87].

1.4.2 Imperfect information flow

The participants in the above excerpt seem to be communicating with each other in floating arena. The question to be asked now is what these two sentences, G_1 and F_2, *mean*, where the meta-linguistic expression is used. The meaning of G_1,

`rotenburo-tte arimasu?` (*do you have hot wells?* `meta`),

cannot be equivalent with that of another sentence,

`rotenburo-wa arimasu?` (*do you have hot wells?* `topic`).

Similarly, the meaning of F_2,

`demo iwaburo-tte-iu ofuro-wa aru`
(*But I have wells called hot springs.* `meta`),

is different from that of another sentence,

`demo iwaburo-wa aru` (*But I have hot springs.* `topic`).

What makes these differences?

The difference must be in the speaker's uncertainty to the given items in the hearer's map. But how can we capture the meaning of sentences uttered in the floating arena? To this question, our answer is given from an operational point of view: The information flow by the utterance is *imperfect*. The notion has been proposed in Channel Theory and has been studied in detail. When their common ground is firmly fixed, linguistic channels work perfect; There is no fear of misunderstanding. One can listen to the other properly, can understand the meaning of sentences, and can figure out what they denote. Channels may be imperfect, however; One may not hear the other properly, may not know the meaning, or may not figure out what they denote. In the above excerpt, the channel works imperfect because it is unsure if they can jointly identify the items uttered by them.

The advantage of adopting the notion, *imperfect* information flow, is that we do not need to throw out all the information we have obtained from utterances when communication *partly* fails. Utterances can still contribute to classifying and updating their common ground even the information flow involved is imperfect. Such a view fits our intuition as to how natural language works in communication.

Another good point of operational account is that it is harmonious with model-theoretic semantics. What has been investigated in semantics of natural language is all *perfect*

cases, where we do not have to care about the imperfectness. Considering imperfect cases can extend the field of study, and we we may study operational aspect of language. The move however does not conflict with model-theoretic approach. We will show through the study of Situation Semantics that formal semantic theories could be built up on our operational semantics by regarding semantic objects as specifying properties of processes. Our work therefore should be regarded as an extension of formal semantics rather than as a completely new approach.

1.4.3 Imperfect information flow in English

We can apply the idea of perfect/imperfect information flow to English as well. Figure 1.12 shows an excerpt from the HCRC Map Task Corpus. Here the giver has *"trig point"* in his map, while the follower does not. At G_4, the giver introduces the item, *"trig point"*, asking the follower if he has it in his map. After realizing he has not it, through F_4 to F_5, the giver tries to persuade the follower to assume there is a trig point in his map at G_6. Interestingly, he first refers to the item as a definite description, *"the tri(g point)"*, then repairs his utterance, changing it to an indefinite description, *"a trig point"*. We can observe in this repair that the imperfectness of their communication can certainly affect one's utterance.

The meaning of indefinite descriptions are accounted for as introducing a new discourse marker according to dynamic semantics [GS91b]. Given the claim is true, what implication can be induced when a speaker utters an indefinite description repeatedly as is observed in the excerpt? The claim faces a difficulty; the speaker is regarded to be introducing a new discourse marker each time he utters *"a trig point"*. Our answer is that the claim is true under the condition that the information flow is perfect, but it does not hold when the information flow is imperfect. By distinguishing these two modes, we can retain the meaning of indefinites as introducing a discourse marker and can explain why he repeats the noun phrase many times.

The approach of introducing the perfect/imperfect flow to semantics enables us to define a meaning for incomplete utterances such as the one observed in G_8. The giver is trying to grade-up the status of *"trig point"* from indefinite to definite description, but he is hesitating to do so, recognizing the follower's difficulty to accept the item as an established one between them. Such an expression as *"the ... "* has not been considered as target of study in semantics, but we can capture the meaning in terms of the information flow.

G$_1$: Right, so you're gonna almost follow the course of that right ... that west lake to the east, the east side of it ... you're gonna follow the *c* the course of it.

F$_1$: Just ... just coming in at the base of it, then?

G$_2$: Yeah, just at the base, and then follow the course of it round sort of a *u*, an *s* shape.

F$_2$: So is it a very very big wide *u* so

G$_3$: Yeah.

F$_3$: I'm gonna avoid *som* ... so that ... I'm avoiding something at the

G$_4$: Well you've got a trig point, do you have a trig point?

F$_4$: No.

G$_5$: No.

F$_5$: Nope.

G$_6$: Right, so in between your west lake and the *tri* There's a trig point, right?

F$_6$: Right.

(after 17 utterances)

G$_7$: So in between it would be ... It would be in between the farmed land and I have a trig point.

F$_7$: Okay fine, right.

G$_8$: Right, okay. So it's just above the ... just at the ... just before the tip of that west lake, but just at the east *co* ... just at that east coast, right?

F$_8$: Okay, fine right, yeah. *Uh-huh.*

Figure 1.12. An excerpt from the HCRC Map Task Corpus

1.5 Related work

1.5.1 Situation Theory and Channel Theory

Situation Theory and Channel Theory are our starting point and form a basis of our theory. The project can be regarded as an enterprise to base those theories on a computational ground with an application in mind, natural language processing. The project is motivated by the result of another project to analyze the meaning of the Japanese particle in dialogue with Situation Theory and Channel Theory. The idea of linguistic channels came out through the investigation of the particle, but machinery provided by those theories was not enough to implement a natural language processing system to test the idea against real data, which have led the author to process calculi.

The project is by no means the first attempt to base Situation Theory on a computational ground. There exist two other projects: ASTL [Bla92] and PROSIT [NPS91]. ASTL is a computer language whose basis is in Situation Theory. The language implements most important features of Situation Theory: individuals, relations, parameters, situations, situation types, propositions, and constraints. It also offers a set of inference rules to prove propositions about a system of propositions and constraints. PROSIT is a knowledge representation language based on Situation Theory and offers a representation of individuals, relations, parameters, situations and constraints. The language however provides for more powerful facilities such as forward and backward search by rules, which is not part of Situation Theory. It is fair to say the language is partly based on Situation Theory but is equipped with other facilities useful as a knowledge representation language. Compared with PROSIT, ASTL is a more conservative implementation of Situation Theory, and its intended application is natural language processing rather than knowledge representation.[6]

Our project is benefited from the results of these projects in considering computational issues related to Situation Theory. An important element of the recent version of Situation Theory which is missing in those languages, however, is *abstraction* and *anchoring*. The introduction of the apparatus is motivated to model Discourse Representation Structures (DRSs). In the research on the semantics of natural language, Discourse Representation Theory (DRT) [Kam84, KR93] has been playing an important role, and a number of researchers are trying to base the theory to a computational ground. Situation Theoretic DRT [Coo93a, Coo93b] is one of such attempts, where DRSs are regarded as an abstraction of situations. Computationally, the operation can be studied in relation with the λ abstraction. The problem is that parameters in DRSs

[6]There is also a work by Glasbey [Gla90], who extended Robin Cooper's Situation Theory based processing environment in Prolog to implement her theory of tense and aspect.

must be abstracted over and anchored to simultaneously, which goes beyond the facility of the λ-calculus. Confronted with the problem, most researchers are trying to enrich the λ-calculus. The solution proposed in EKN is, for example, to index parameters so that they can be anchored to regardless of the order of substitution operations (§4.2.5 and §4.2.6). Pinkal's compositional DRT [BMM+94] preserves the λ-calculus, but devises a compose function that will do the same job, i.e., managing the anchoring operation so that parameters can be substituted by proper values.

Given our motivation to study linguistic acts, the issue of abstraction and anchoring must be investigated as the most important problem in our project. To base simultaneous abstraction on a computational ground, however, we do not extend the λ-calculus, but choose a process calculus.

1.5.2 Process calculi

Going back to the intuition to regard DRSs as an abstract object, one can argue that DRSs are meant to model a frame of mind, whose slots will be filled with concrete values when embedded into an environment. In this conceptualisation, anchoring can be regarded as a process of filling those slots. Such a process can be captured with the idea of simultaneous abstractions, but it can be modelled more directly as concurrent interactions of an agent with environments [Fuj94]. This conceptualisation fits with a recent development of Situation Theory, too, where indices are not merely natural numbers or utterances, but a more complex object that involves a referring action [Coo93a].

To implement the idea of concurrent interactions, we turn to a process calculus, namely the π-calculus [Mil93b, MPW92]. The calculus is proposed as a model for concurrent processes and can be used to model linguistic acts mentioned above as well, which is the main argument of this thesis. The striking point of the calculus is that it can model the indexed abstractions naturally. The following is, for instance, an explanation on terms of the calculus:

> The first class of terms consists of *guarded terms* $g.P$, where P is a term and g is a *guard*; guards g have the form
>
> $$g ::= \overline{x}\langle y\rangle \mid x(y)$$
>
> Informally, $\overline{x}\langle y\rangle.P$ means 'send the name y along the link named x, and then enact P'; on the other hand, $x(y).P$ means 'receive any name z along the link named x, and then enact $P\{^z/_y\}$'. Thus the guard $x(y)$ is like the λ-prefix λy in that it binds y. ([Mil92c]: p.121)

Thus, we can see a π-calculus term, $x(y)$, as an indexed abstraction over y with x. This correspondence motivates us to look into the calculus in depth for linguistic applications.

It may appear to be odd to model situation theoretic objects based on process calculi, but Situation Theory has been historically benefited from work on process calculi. For instance, one of the motivations of Non-Well-Founded Set Theory [Acz88], which lays a foundation for Situation Theory [AL91], is to define a semantics for Synchronous Calculus of Communicating Systems (SCCS) [Mil83, Mil89], a precursor to the π-calculus. Therefore, it should not be so difficult to model an earlier version of Situation Theory [BE87] with SCCS. But we are here interested in exploring the π-calculus to model more recent versions of Situation Theory.

It has been shown that with the π-calculus we can encode various data structures such as list [Mil93b, MPW92], the λ-calculus [Mil92c], and a concurrent object-oriented language [Wal95]. Our project is benefited from these encodings and aims to apply the technique to computational linguistics.

1.5.3 Dynamic semantics

In connection with DRT, there are several projects for reworking the theory in logical setting, which are often termed as *dynamic semantics* (e.g., [GS91a, GS91b, Mus92, Mus93]). The diversity of those projects makes it difficult to compare our work with them in detail, but we take the same approach in formalising ideas, program logic. The difference is in the selection of models.

To clarify our relationship with those projects, we examine the relation between program and logic. The aim of *dynamic logic* [Pra81], a source of insight for dynamic semantics, is to characterise programs and infer its properties logically, e.g., what value will be assigned to a variable when an instruction is executed. Let P be a program denoting a function or relation, and φ a formula. A dynamic logic formula is expressed as:

$$[\,P\,]\varphi$$

where the program is embedded in the formula as *modal operators*. The meaning of the expression is that the state of machine after executing P is characterised as φ. A Hoare-style formula too can be categorised into this formalism since $\{\varphi\}P\{\psi\}$, the state changes from φ to ψ when P is executed, can be translated to $\varphi \models [\,P\,]\psi$.

In dynamic semantics, roughly speaking, the programs are conceived of as an instruction that can change the state of memory of computer by assigning a value to a variable. The meaning of the program is then captured as the change of memory states upon the execution of interactions, say $\varphi \rightarrow \psi$. Additional conditions specifying relations between values or their properties make it possible to express complex structures, e.g., DRSs. Comparing such a conceptualization with ours, our notion of program is different. In our study, programs are several processes that run concurrently and can interact with each other to exchange data. The different programs mean that our model is different from those for dynamic semantics.

A strategic difference lies in the way to characterise programs logically. Following Hennessy-Milner logic [HM85], we will characterise programs by describing the properties pertaining to them:

$$P \models \varphi$$

where P is a program and φ is a formula. These two characterisations are, however, not so different from each other as their appearances suggest. It has been known that the former is an alternative formulation of a meta-language to specify programs [Abr91].

In dynamic semantics, the idea of concurrency is recently considered to temper the problem of nondeterministic computation [Fer93]. The solution is to preserve all alternatives with disjoint union when a source of nondeterminism is encountered. In this model, however, the idea of communication is still missing. Therefore, the work is not particularly related with ours.

1.5.4 Concurrent natural language processing

Here is a note on the difference between *parallel* and *concurrent*. While parallel architectures for computation are studied in computer science, the notion of *concurrency* is an independent concept of parallel processing. Parallelism is to do with execution on hardware or very low level software architectures, whilst concurrency concerns with design of systems or programs.[7]

As for concurrent natural language processing, the most relevant model in spirit with ours is proposed by Bröker, Hahn, and Schacht [SHB94, BHS94, BSSH94]. In their *ParseTalk* model, parsing is modelled as message passing based on Hewitt's actor model [Agh86]. Grammar rules are not separated from lexicon, but encoded into each lexical item. Such a lexical item is encoded as an actor, and actors can communicate with each other to parse sentences. Our project share the intuition to model lexical items as autonomous agents, but the level of modelling is different. While their model is defined at a higher level, say actor model, ours is defined at more primitive level, say process algebra. Our advantage is that we can model feature structures and other data structures in the same framework as the model is so fine grained, while their model forces to combine different frameworks, e.g., introducing feature structures in separate form. But the point should not be exaggerated; the aim of the project is to build a practical system while ours to investigate a concurrent theory for NLP. Our model

[7] I owe to Gert Smolka in realising this.

should serve as a basis to analyse and compare various concurrent NLP models such as *ParseTalk* model in broader contexts. Another difference is their emphasis on general knowledge and ambiguity. Class hierarchy is, for instance, employed to disambiguate syntax. That sort of issue is not considered in this thesis.

Oz is a concurrent constraint programming language developed by Gert Smolka and his group at DFKI (German Research Center for Artificial Intelligence) [Smo94b]. The language combines ideas from the π-calculus with constraint programming and can find a solution satisfying constraints effectively owing to concurrency. The emphasis is on the design of concurrent constraint language, but application to NLP is planned. One way to encode feature structures as concurrent objects has been proposed by Smolka [Smo94a] and a bottom-up parser is implemented as a demo program. The facilities provided by the language is so rich that it seems that anything can be encoded with it. The difference in strategy and objectives, however, should be pointed out with our project. While the language is rich in facilities, we start with primitive elements of computation. The complex behaviour of systems will be modelled through combining small building blocks. This is because we use the π-calculus as a model to study the meaning of concurrent objects. The simpler the model is, the easier to study its properties. We are interested in identifying the least set of operations and how complex objects can be built up using them. The Oz language is on the other hand designed as a practical programming language to build systems. Therefore, they cannot be compared with each other. Our results should, however, contribute to building NLP systems using Oz language.

1.6 The aim of the thesis and overview

1.6.1 The aim of thesis

The aim of the thesis is to construct linguistic channels as a system of communicating processes. The ultimate goal is to answer the three questions: (1) how discourse can interact with the parsing phase, (2) how plans and goals can interact with the interpretation phase, and (3) how circumstances can interact with the evaluation phase. But there are many things to be done towards the goal. The first step is to show how we can model linguistic channels as a system of communicating processes, while respecting the two desiderata: (a) sentence and situation types must be organised and represented in the way that they can interact with discourse, plans and goals, and circumstances at any point, and (b) discourse, plans and goals, and circumstances, must be represented in the way that the phases of parsing, interpreting, and evaluating utterances can interact with them at any point. Once we succeed in the project, we can go forward to empirical studies on how contexts can influence the phases of language understanding in detail through simulation.

In the thesis, we will turn to the π-calculus to describe systems of communicating processes. The project may sound unusual, but we do *not* intend to invent a completely new linguistic theory. We will rather reinterpret conventional theories in the light of concurrency and communication. We will introduce, for example, feature structures [PS94] to express syntactic types and Extended Kamp Notation (EKN) [BC91, BC93] to express situation types. These notations will serve as constraints to specify possible class of computations. In this respect, the project can be seen as defining an operational semantics to those remodelled theories, but one should not be misguided by the side effects. We are interested in modelling the information flow by utterances as a system of communicating processes in order to study roles that contexts can play in language use.

1.6.2 Overview

The thesis is organised as follows:

Chapter 2 We first explain the π-calculus. Since the calculus, and probably the idea of communication and concurrency, too, can be unfamiliar to the reader, the explanation is given in detail and self-contained so that the succeeding chapters can be understood without any prior knowledge of process calculi.

Chapter 3 We investigate how logic can be used to specify mobile processes. We turn to Combinatorial Intuitionistic Linear Logic and show how processes defined in the π-calculus and a molecular language can be specified in the logic.

Chapter 4 We construct semantic objects employed in Situation Semantics in the π-calculus. The result obtained in the previous chapter is used to relate our study on Situation Semantics with Channel Theory.

Chapter 5 We encode a concurrent chart parser in the calculus. We start by encoding feature structures and then encode a parser that analyses sentences as a result of interactions between agents. We show how a semantic representation can be constructed by the parser.

Chapter 6 We conclude the thesis by comparing our framework with other approaches.

Chapter 2

A Calculus of Interaction

The aim of this chapter is to give the reader an insight into concurrent computation and communication. We start by explaining the π-calculus because our modelling is based on it. Through examination of its underlying computation, we investigate what concurrent computation and communication are and how they can be controlled. The investigation enables us to identify the minimum set of elements we need to define systems of communicating processes.

2.1 The π-calculus

2.1.1 From simple automata to systems of communicating processes

The section introduces the reader to the π-calculus [MPW92, Mil93b]. We start by presenting simple automata, then will enrich them till we obtain the polyadic π-calculus [Mil93b]. The section firstly aims to give the reader basic ideas on the calculus, then we define its syntax and semantics formally in §2.1.4. In what follows, capital letters, A, B, and C range over automata or agents, P, Q, and R over processes, and italics $a, b, \cdots$, over actions. The terminology of automata or agent is used to indicate entities modelled as processes. Processes are defined with respect to actions.

Simple automata

Let A be an automaton that accepts a sequence of symbols, $\langle a, b\rangle$, and terminates (Figure 2.1). We define the initial state of A, P_0, by its possible courses of actions, e.g. accepting a and b, then terminating. We express such a state as

$$P_0 =_{\mathrm{def}} a.b.\mathbf{0}$$

where **0** means termination[1]. After accepting the symbol a, the state of A turns into another state, P_1, such as accepting b and terminating, written as

$$P_1 =_{\text{def}} b.\mathbf{0}$$

To express the change of the state caused in A by accepting a, we write

$$P_0 \xrightarrow{a} P_1$$

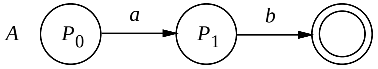


Figure 2.1. A simple automaton

Simple communicating systems

Simple automata can only *accept* symbols, but we can think of another sort of action, *emitting* symbols. A system of automata which can interact with each other by emitting and accepting symbols forms a communicating system. Let the initial state of A be P_0 as before and the initial state of another automaton B, Q_0, such as emitting the sequence of symbols, $\langle a, b\rangle$ (Figure 2.2). We distinguish such an output action with overline and write the definition as

$$Q_0 =_{\text{def}} \overline{a}.\overline{b}.\mathbf{0}$$

Suppose a communicating system is composed of A and B, whose initial state is defined as $P_0 \mid Q_0$, meaning that P_0 and Q_0 are concurrently active. Because they are active, they can communicate, e.g., by B's emitting a symbol and A's accepting it. Suppose now B emits a and A picks it up. Upon this interaction, the state of the system turns into another state, which is expressed as

$$P_0 \mid Q_0 \xrightarrow{\tau} P_1 \mid Q_1$$

where P_1 is as before and $Q_1 =_{\text{def}} \overline{b}.\mathbf{0}$. The symbol, '$\tau$', read as *silent* action, means the interaction between A and B is invisible from outside. Hereinafter, we may write $\rightarrow$ for a finite sequences of $\xrightarrow{\tau}$ for readability.

[1]Hereinafter, **0** may be omitted for readability.

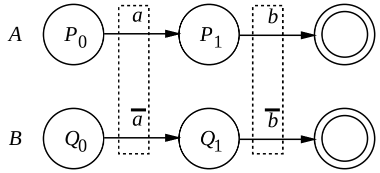


Figure 2.2. A simple communicating system

Data communicating systems

The interaction between A and B can be regarded as establishing a communication channel or port through which data can be exchanged. In the figure 2.3, we depict the agents, A and B, as circles with bold line and channels between them, a and b, as lines whose ends are emphasised with squares.[2]

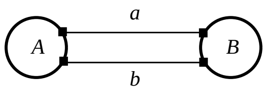


Figure 2.3. A channel

Now suppose A is willing to receive a number from B through a port c and B wants to emit '5' through it (Figure 2.4). Let P_{10} be A's initial state, defined as

$$P_{10} =_{\text{def}} c(x).P_{11}$$

where x is a parameter to be substituted for upon the interaction, whereas B's initial state, Q_{10}, is defined as

$$Q_{10} =_{\text{def}} \overline{c}\langle 5\rangle.Q_{11}$$

where '5' is the number to be emitted.[3] Upon the interaction, the number '5' is emitted

[2]It may be confusing that we use the same alphabets, a and b, to indicate both the actions establishing channels and channels established by them. We may adopt a different font for channels, e.g., **a** and **b**, to avoid the confusion, but we follow in the thesis the convention adopted in the literature [Mil89].

[3]One may wonder why $\overline{c}\langle 5\rangle$ is not expressed simply as $\overline{c}(5)$. There is a historical reason for this. In some versions of the calculus, $\langle\rangle$ is used to indicate the name is free, while $()$ to indicate it is bound. In fact, there were the notions of free input, e.g., $c\langle x\rangle$, and bound output, e.g., $\overline{c}(5)$, as well as bound input and free output explained here. See §2.1.2 to know about the notions of free and bound names.

to A, and x is substituted by it, which we write as

$$P_{10} \mid Q_{10} \rightarrow P_{11}\{{}^{5}/_{x}\} \mid Q_{11}$$

The expression, $P_{11}\{{}^{5}/_{x}\}$, means any x appeared in P_{11} to be substituted by '5'.

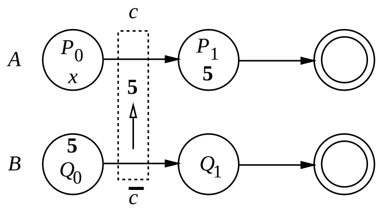


Figure 2.4. A data communicating system

The monadic π-calculus

We have so far distinguished ports from data, but we can treat them uniformly as *names*, which enables us to model *mobility*: an access to a port can be exchanged, too. Suppose B needs to emit '5' to C and wants A to take over the job. Suppose also C is accessible only through a port a, and A and B communicates with only through another port b. Assume first A would have an access to a (Figure 2.5). Then, what B needs to do is to send A '5' so that it can emit the number to C. A's state P_{20} may be defined as $b(x).\overline{a}\langle x\rangle.P_{21}$, B's state Q_{20} as $\overline{b}\langle 5\rangle.Q_{21}$, and C's state R_{20} as $a(y).R_{21}$. Upon B's emitting '5' to A and its emitting it to C, the state of the system would change as follows:

$$\begin{array}{llll}
 & A & B & C \\
 & b(x).\overline{a}\langle x\rangle.P_{21} & \mid \overline{b}\langle 5\rangle.Q_{21} & \mid a(y).R_{21} \\
 & \text{(B emits 5 to A via b.)} & & \\
\rightarrow & \overline{a}\langle 5\rangle.P_{21}\{{}^{5}/_{x}\} & \mid Q_{21} & \mid a(y).R_{21} \\
 & \text{(Receiving it, A substitutes it for x, and emits it to C via a.)} & & \\
\rightarrow & P_{21}\{{}^{5}/_{x}\} & \mid Q_{21} & \mid R_{21}\{{}^{5}/_{y}\} \\
 & \text{(C receives 5 and substitutes it for y.)} & &
\end{array}$$

Now, suppose that A does not have the access to C, the port a (Figure 2.6). How can B pass the job to A? Since A does not have the access, B has to first emit it to A,

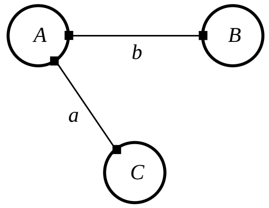


Figure 2.5. Monadic π-calculus: motivation

which is made possible in the π-calculus. B first sends A the port name a via b, then the number. Let A's initial state P_{30} be $b(x).b(z).\overline{x}\langle z\rangle.P_{31}$, B's state $Q_{30} =_{\mathrm{def}} \overline{b}\langle a\rangle.\overline{b}\langle 5\rangle.Q_{31}$, and C's state $R_{30} =_{\mathrm{def}} a(y).R_{31}$. The state of the system will change upon interactions as follows: [4]

$$\begin{array}{llll}
 & A & B & C \\
 & b(x).b(z).\overline{x}\langle z\rangle.P_{31} & \mid \overline{b}\langle a\rangle.\overline{b}\langle 5\rangle.Q_{31} & \mid a(y).R_{31} \\
 & \text{(B emits a to A via b.)} & & \\
\rightarrow & b(z).\overline{a}\langle z\rangle.P_{31}\{{}^{a}/_{x}\} & \mid \overline{b}\langle 5\rangle.Q_{31} & \mid a(y).R_{31} \\
 & \text{(A substitutes it for x. B emits 5 to A via b.)} & & \\
\rightarrow & \overline{a}\langle 5\rangle.P_{31}\{{}^{a}/_{x},{}^{5}/_{z}\} & \mid Q_{31} & \mid a(y).R_{31} \\
 & \text{(A substitutes it for z and emits it to C via a.)} & & \\
\rightarrow & P_{31}\{{}^{a}/_{x},{}^{5}/_{z}\} & \mid Q_{31} & \mid R_{31}\{{}^{5}/_{y}\} \\
 & \text{(C substitutes 5 for y.)} & &
\end{array}$$

The polyadic π-calculus

In the polyadic version, agents can exchange sequences of data, not only a single item, upon a single interaction. Suppose, for instance, A wants to receive a sequence of three items, while B wants to emit a sequence, $\langle 4,5,6\rangle$, where they are linked through a port d. Let the A's state P_{40} be $d(x,y,z).P_{41}$ and the B's state $Q_{40} =_{\mathrm{def}} \overline{d}\langle 4,5,6\rangle.Q_{41}$. The state of the system would change upon a single interaction as follows:

$$\begin{array}{ll}
A & B \\
d(x,y,z).P_{41} & \mid \overline{d}\langle 4,5,6\rangle.Q_{41}
\end{array}$$

[4] In the transition, the name x standing for a port name in $b(z).\overline{x}\langle z\rangle.P_{31}$ is substituted by a after the first interaction between A and B.

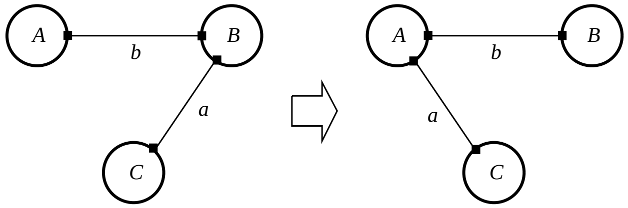


Figure 2.6. Monadic π-calculus: mobility

$$\rightarrow \quad P_{41}\{{}^{4}\!/_{x}, {}^{5}\!/_{y}, {}^{6}\!/_{z}\} \qquad \mid Q_{41}$$

2.1.2 Free and bound names

In the above examples, we assumed that all names are *free*, that is, accessible from any processes. However, as is the case in logic, introducing the distinction between *free* and *bound* names is useful to describe communication between processes. In the π-calculus when a name is *bound*, only the processes within its scope can get access to it. This enables one to define *local* names analogous to the local variables in procedural computer languages or provides one with a similar function to data-hiding in object oriented languages. The name-passing ability of the calculus brings about interesting phenomena, *scope intrusion* and *scope extrusion*, which is explained in this section.

Definition

The free and bound names are defined as is shown in table 2.1. The names that will be substituted by another names, e.g., y in $x(y)$, are bound while the names emitted are not. The names used for communication, e.g., x in $x(y)$ or $\overline{x}\langle y\rangle$, are free unless they are bound explicitly. The restriction operator, ν, read *"new"*, can bind any name in its scope. The name y in $(\nu\, y)(\overline{x}\langle y\rangle)$ is, therefore, bound. We will utilise bound names in chapter 5 to encode a concurrent chart parser, where we model channels between adjacent agents as bound ones, the agents that are assigned to words or phrases of sentences.

Passing channels through private channels

When a name is bound, it is accessible only among the processes within the scope. In the example below, b is private between A and B, not accessible from C.

expressions	bound	free
$x(y)$	$\{y\}$	$\{x\}$
$\overline{x}\langle y\rangle$	$\emptyset$	$\{x, y\}$
$(\nu\, y)(\overline{x}\langle y\rangle)$	$\{y\}$	$\{x\}$
$(\nu\, x)(\overline{x}\langle y\rangle)$	$\{x\}$	$\{y\}$
$(\nu\, x, y)(\overline{x}\langle y\rangle)$	$\{x, y\}$	$\emptyset$
$(\nu\, x)(x(y))$	$\{x, y\}$	$\emptyset$

Table 2.1. Free and bound names

$$\begin{array}{llll}
 & \quad A & \quad B & \quad C \\
 & (\nu\, b)(b(x).b(z).\overline{x}\langle z\rangle.P & \mid \overline{b}\langle a\rangle.\overline{b}\langle 5\rangle.Q) & \mid a(y).R \\
\rightarrow & (\nu\, b)(b(z).\overline{a}\langle z\rangle.P\{{}^{a}\!/_{x}\} & \mid \overline{b}\langle 5\rangle.Q) & \mid a(y).R
\end{array}$$

Graphically, private channels are depicted as a link whose label is put on at its both ends within the process, not in the middle (Figure 2.7).

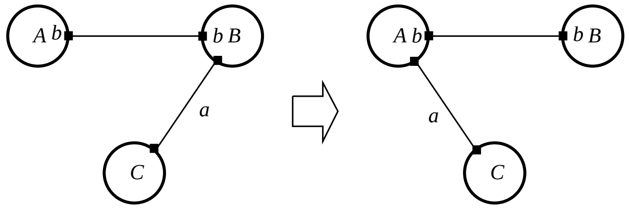


Figure 2.7. Passing channels through private channels

Scope intrusion

When a process receives a channel whose name conflicts with its private channel, the private one will be renamed to avoid confusion. In the following example, the private channel, a, between A and D will be renamed to a' when A receives another channel from B, the channel which happens to bear the same name a (Figure 2.8).

$$\begin{array}{lllll}
 & \quad D & \quad A & \quad B & \quad C \\
 & (\nu\, a)(\ a(v) & \mid b(x).b(z).\overline{x}\langle z\rangle.\overline{a}\langle z\rangle.P\ ) & \mid \overline{b}\langle a\rangle.\overline{b}\langle 5\rangle.Q & \mid a(y).R \\
\rightarrow & (\nu\, a')(\ a'(v)\{{}^{a'}\!/_{a}\} & \mid b(z).\overline{a}\langle z\rangle.\overline{a'}\langle z\rangle.P\{{}^{a}\!/_{x}\}\{{}^{a'}\!/_{a}\}) & \mid \overline{b}\langle 5\rangle.Q & \mid a(y).R
\end{array}$$

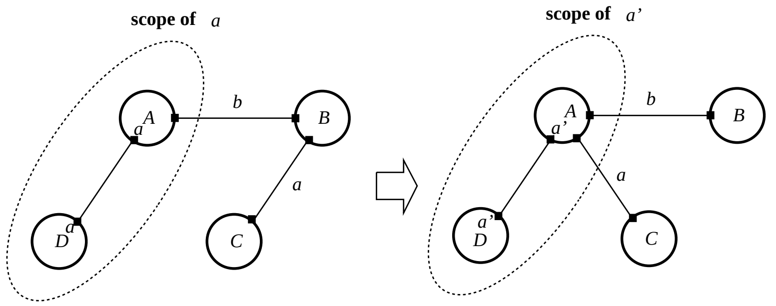


Figure 2.8. Scope intrusion

Scope extrusion

When a process emits a private channel, its scope is extruded to the receiving process. In the example (Figure 2.9), the scope of a private between B and C is extruded to A when it is passed to A through b. (There is a familiar example in the C programming language, where a module passes another an address of or pointer to its local variable so that it can read or write the value of the variable.)

$$\begin{array}{llll} & A & B & C \\ & b(x).b(z).\overline{x}\langle z\rangle.P & \mid\ (\nu\, a)(\overline{b}\langle a\rangle.\overline{b}\langle 5\rangle.Q & \mid\ a(y).R) \\ \rightarrow & (\nu\, a)(b(z).\overline{a}\langle z\rangle.P\{^a/_x\} & \mid\ \overline{b}\langle 5\rangle.Q & \mid\ a(y).R) \end{array}$$

2.1.3 Other operators

In the calculus, there are another three operators, *replication*, *match*, and *choice*.

Replication

The replication operator, !, replicates the process finitely many times. For example, a process definition, $!\,\overline{c}\langle 5\rangle$, means finitely many copies of $\overline{c}\langle 5\rangle$:

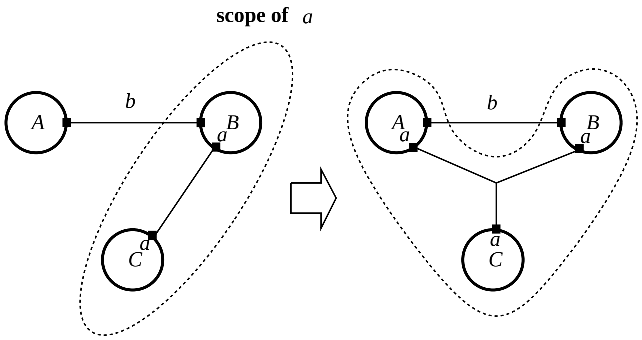


Figure 2.9. Scope extrusion

$$\overline{c}\langle 5\rangle \mid \overline{c}\langle 5\rangle \mid \overline{c}\langle 5\rangle \mid \cdots \mid \overline{c}\langle 5\rangle$$

Typically, the replication operator is used to allow a single process to provide data to all processes that request it. Observe, for example, the following case:

$$\overline{c}\langle 5\rangle \mid c(x).P \mid c(y).Q$$

This system may change into two different states. If the second process, $c(x).P$, interacts with the first, $\overline{c}\langle 5\rangle$, the transition is shown as:

$$\overline{c}\langle 5\rangle \mid c(x).P \mid c(y).Q \rightarrow P\{^5/_x\} \mid c(y).Q$$

while it changes into different state if the third, $c(y).Q$, interacts with it:

$$\overline{c}\langle 5\rangle \mid c(x).P \mid c(y).Q \rightarrow c(x).P \mid Q\{^5/_y\}$$

If the first process, $\overline{c}\langle 5\rangle$, is replicated twice, both processes can get the number. That is, if we define the first process as:

$$!\,\overline{c}\langle 5\rangle \mid c(x).P \mid c(y).Q$$

then it can provide for the number twice:

$$!\,\overline{c}\langle 5\rangle \mid \overline{c}\langle 5\rangle \mid \overline{c}\langle 5\rangle \mid c(x).P \mid c(y).Q$$

therefore, both processes can get 5 as is expected:

$$!\,\overline{c}\langle 5\rangle \mid \overline{c}\langle 5\rangle \mid \overline{c}\langle 5\rangle \mid c(x).P \mid c(y).Q \rightarrow\, !\,\overline{c}\langle 5\rangle \mid P\{^5/_x\} \mid Q\{^5/_y\}$$

Replication and bound names

The combination of the scope of bound names and replication may complicate the calculus slightly. Imagine what effects the following system brings about:

$$!\,\overline{c}\langle 5\rangle \mid !\,c(x).\mathbf{0}$$

The effect is not just a finite number of substitutions of a single name x by 5, but the number will substitute finitely many distinct x_i because the system can be presented as:

$$!\,\overline{c}\langle 5\rangle \mid c(x).\mathbf{0} \mid c(x).\mathbf{0} \mid c(x).\mathbf{0} \mid c(x).\mathbf{0} \mid \cdots \mid c(x).\mathbf{0}$$

which is alphabetically equivalent to:

$$!\,\overline{c}\langle 5\rangle \mid c(x_1).\mathbf{0} \mid c(x_2).\mathbf{0} \mid c(x_3).\mathbf{0} \mid c(x_4).\mathbf{0} \mid \cdots \mid c(x_n).\mathbf{0}$$

where each x_i is a distinct name as they are bound by different binding occurences. On the other hand, in the following system the number will substitute a single name, x, finitely many times as each x is within the scope of the preceeding xs:

$$!\,\overline{c}\langle 5\rangle \mid c(x).c(x).c(x).c(x)\cdots c(x).\mathbf{0}$$

The same is true for output actions. While each x receives the same number, 5, in the above example, they will receive different names if the emitted datum is a bound name:

$$!\,(\nu\, a)\overline{c}\langle a\rangle \mid c(x).c(x).c(x).c(x)\cdots c(x).\mathbf{0}$$

Since the name a is bound, the action $\overline{c}\langle a\rangle$ emits a distinct name, a_i, each time it emits a datum. Therefore, x will be substituted by different names each time the input action, $c(x)$, is performed.

This will change however if the scope of restriction, $(\nu\, a)$, is outside that of replication, (!),

as follows:

$$(\nu\, a)(\ !\,\overline{c}\langle a\rangle\ ) \mid c(x).c(x).c(x).c(x)\cdots c(x).\mathbf{0}$$

Since a created by $\overline{c}\langle a\rangle$ is the same each time, the input actions, $c(x)$ will replace the identical a for x each time it receives it.

Match

The match operator restricts the performance of actions by referring to substitution environments. In the following system, for example, the formulas, $[x=5]$ and $[x=3]$, allow their operant actions to be active only when they are satisfied. Therefore, while both actions can be performed if they were not restricted with, only the action, $\overline{a}\langle d\rangle$, is possible when the name x is substituted by 5 through c:

$$\begin{array}{rllll} & \overline{c}\langle 5\rangle.\mathbf{0} & \mid c(x). & (\ [x=5]\ \overline{a}\langle d\rangle.\mathbf{0} & \mid [x=3]\ \overline{b}\langle e\rangle.\mathbf{0}) \\ \rightarrow & \mathbf{0} & \mid & [x=5]\ \overline{a}\langle d\rangle.\mathbf{0}\{^5/_x\} & \mid [x=3]\ \overline{b}\langle e\rangle.\mathbf{0}\{^5/_x\} \\ \xrightarrow{\overline{a}\langle d\rangle} & \mathbf{0} & \mid & \mathbf{0} & \mid [x=3]\ \overline{b}\langle e\rangle.\mathbf{0}\{^5/_x\} \end{array}$$

Choice

The choice operator, +, ensures only one action among more than one candidates is performed and eliminates others. It is used typically combined with the match operator. In the following example, the action $\overline{b}\langle e\rangle$ will be eliminated when the other action, $\overline{a}\langle d\rangle$, is chosen to perform.

$$\begin{array}{rllll} & \overline{c}\langle 5\rangle.\mathbf{0} & \mid c(x). & (\ [x=5]\ \overline{a}\langle d\rangle.\mathbf{0} & +\ [x=3]\ \overline{b}\langle e\rangle.\mathbf{0}) \\ \rightarrow & \mathbf{0} & \mid & [x=5]\ \overline{a}\langle d\rangle.\mathbf{0}\{^5/_x\} & +\ [x=3]\ \overline{b}\langle e\rangle.\mathbf{0}\{^5/_x\} \\ \xrightarrow{\overline{a}\langle d\rangle} & \mathbf{0} & \mid & \mathbf{0} & \end{array}$$

Note the operator may make transitions nondeterministic although it is useful to exclude unneeded actions. See, for example, a system such as

$$\overline{c}\langle 5\rangle.\mathbf{0} \mid c(x).(\ \overline{a}\langle d\rangle.\mathbf{0} + \overline{b}\langle e\rangle.\mathbf{0})$$

is nondeterministic as it is unpredictable which action, either $\overline{a}\langle d\rangle$ or $\overline{b}\langle e\rangle$, will be performed after the name x is substituted by 5.

2.1.4 Formal definition

Now that we have shown how the π-calculus models processes, we define the full calculus formally.

Syntax

The syntax is defined as follows, where P and Q range over *processes*, and α a *particle*. Also x and y are names, and $\vec{x}$ and $\vec{y}$ vectors of names.

$$
\begin{array}{lll}
P \longrightarrow & \mathbf{0} & \text{(termination)} \\
& \mid \alpha & \text{(particle)} \\
& \mid \alpha \,.\, Q & \text{(prefix)} \\
& \mid (\nu\, x)P & \text{(restriction)} \\
& \mid [x = y]P & \text{(match)} \\
& \mid\, !\, P & \text{(replication)} \\
& \mid P \mid Q & \text{(parallel)} \\
& \mid P + Q & \text{(choice)} \\
\alpha \longrightarrow & \overline{x}\langle\vec{y}\rangle & \text{(exporting)} \\
& \mid x(\vec{y}) & \text{(importing)} \\
& \mid \tau & \text{(silent action)}
\end{array}
$$

Semantics

The semantics is defined with the set of reduction rules over processes and the *structural congruence* relation between them. The *reduction* relation is the smallest relation satisfying the rules shown in the figure 2.10.[5] The structural congruence relation, which is referred in STRUCT, is defined as is shown in the figure 2.11. The structural congruence relation is useful to avoid tedious definition of reduction relation. The combination of STRUCT and the structural congruence relation, for example, can deduce many other reduction rules. In the definition RES, $n(\alpha)$ means all names comprising α. In the definition of PAR, $bn(\alpha)$ means the bound names in α, while $fn(Q)$ means the free names in Q.

To illustrate how the semantics works[6], observe the following example of a vending machine[7]. Suppose there is a vending machine that sells a big chocolate bar for a 2p coin and a little one for a 1p coin when pressed a button, big or little. Such a machine, depicted in the figure 2.12, can be defined as:

$$
\begin{aligned}
Ven &=_{\text{def}} \;\; \texttt{2p}.Ven_b \;+\; \texttt{1p}.Ven_l \\
Ven_b &=_{\text{def}} \;\; \texttt{big}.\texttt{collect}_b.Ven \\
Ven_l &=_{\text{def}} \;\; \texttt{little}.\texttt{collect}_l.Ven
\end{aligned}
$$

where each importing action such as `2p` receives no name upon interactions, but is just performed. Note also we do not care about who performs which action, but its behaviour as a system composed of the vendor machine and buyers. Getting a coin, `2p` or `1p`, and

[5] PRE stands for *"prefix"*, COM for *"communication"*, RES for *"restriction"*, REP for *"replication"*, PAR for *"parallel"*, SUM for *"summation"*, and STRUCT for *"structural congruence"*.

[6] This approach, advocated by Plotkin [Plo81], is called Structural Operational Semantics. The approach is syntactic, not model-theoretic.

[7] The example is taken from [Mil89]

PRE: $\alpha.P \xrightarrow{\alpha} P$

COM: $\dfrac{P \xrightarrow{\overline{a}\langle y\rangle} P' \quad Q \xrightarrow{a(x)} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'\{y/x\}}$

RES: $\dfrac{P \xrightarrow{\alpha} P'}{(\nu\, y)P \xrightarrow{\alpha} (\nu\, y)P'} \quad y \notin n(\alpha)$

MATCH: $\dfrac{P \xrightarrow{\alpha} P'}{[a = a]P \xrightarrow{\alpha} P'}$

REP: $\dfrac{P \mid\, !\, P \xrightarrow{\alpha} P'}{!\, P \xrightarrow{\alpha} P'}$

PAR: $\dfrac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \quad bn(\alpha) \cap fn(Q) = \emptyset$

SUM: $\dfrac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$

STRUCT: $\dfrac{Q \equiv P \quad P \rightarrow P' \quad P' \equiv Q'}{Q \rightarrow Q'}$

Figure 2.10. Reduction relation for the π-calculus

alpha-convertible:	$P \equiv Q$ whenever P is alpha-convertible to Q
synmetric monoid laws for $\mid$:	$P \mid \mathbf{0} \equiv P$
	$P \mid Q \equiv Q \mid P$
	$P \mid (Q \mid R) \equiv (P \mid Q) \mid R$
synmetric monoid laws for $+$:	$P + \mathbf{0} \equiv P$
	$P + Q \equiv Q + P$
	$P + (Q + R) \equiv (P + Q) + R$
replication:	$!\, P \equiv P \mid\, !\, P$
restriction:	$(\nu\, x)\mathbf{0} \equiv \mathbf{0}$
	$(\nu\, x)(\nu\, y)P \equiv (\nu\, y)(\nu\, x)P$
	$(\nu\, x)(P \mid Q) \equiv P \mid (\nu\, x)Q$ if x is not free in P

Figure 2.11. Structural congruence relation for the π-calculus

pushed a button, `big` or `little`, should be performed by the vending machine, while getting a big or little chocolate bar, $\texttt{collect}_b$ or $\texttt{collect}_l$, should be performed by a person, but it is not explicitly mentioned in this example.

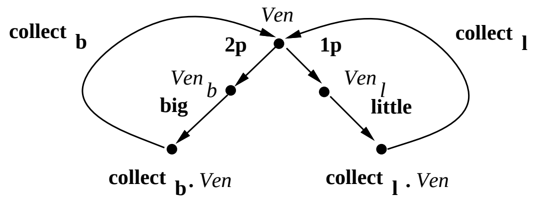


Figure 2.12. The description of a vending machine

Now given the semantics, we can prove that the state of the machine would change from Ven to Ven_b when it gets 2p as follows (the goal comes at bottom):

$$\cfrac{\cfrac{\texttt{2p}.Ven_b \xrightarrow{\texttt{2p}} Ven_b}{\texttt{2p}.Ven_b + \texttt{1p}.Ven_l \xrightarrow{\texttt{2p}} Ven_b}}{Ven \xrightarrow{\texttt{2p}} Ven_b} \qquad \begin{array}{l}\textsc{pre}\\ \text{by } \textsc{sum}\\ \text{by definition}\end{array}$$

The first formula is the axiom, PRE. The step from the first to the second is carried out by applying the SUM rule. The final step is just to rewrite the antecedent to Ven based on the first definition of the vending machine.

2.2 Graphic presentations of processes

We look into the underlying computational mechanism of the π-calculus in depth in order to make it easy to define a language (§3.3).[8] As we believe it helps to grasp the essence of computation intuitively, we devise a graphic notation to discuss computational issues.

[8] It will turn out that the π-calculus is too restrictive in terms of prefixing to model situation-theoretic objects (Chapter 4). We will therefore propose a language $\mathcal{L}_0$ in the next chapter, which is more relaxed in that respect. We can however disregard the intention here.

2.2.1 State transition graphs

An example of graphical presentation is State Transition Graphs (STGs) such as shown in the figure 2.12. The diagram is useful to depict the behaviour of a single process, but not so much when applied to systems of communicating processes because every edge is labelled simply as τ when there are interactions. Suppose there is a system composed of two processes such as:

$$\overline{a}\langle b\rangle.\overline{c}\langle d\rangle \ \mid a(x).c(y)$$

These two processes can communicate with each other by the first process consecutively emitting b and d through a and c, respectively, while the second receiving and substituting them for x and y. Their interactions are, however, invisible from outside:

$$\overline{a}\langle b\rangle.\overline{c}\langle d\rangle.\mathbf{0} \mid a(x).c(y).\mathbf{0} \xrightarrow{\tau} \overline{c}\langle d\rangle.\mathbf{0} \mid c(y).\mathbf{0}\{^a/_x\} \xrightarrow{\tau} \mathbf{0} \mid \mathbf{0}\{^a/_x, {}^b/_y\}$$

As we cannot distinguish these two transitions, STGs are not of much use to investigate interactions between processes.

2.2.2 Dataflow graphs

Confining ourselves to the transaction of names at the moment, we will depict the dependency between names respecting to name-passing as a graph whose nodes are names and edges actions operating on them, exporting or importing.

Buffer and channels

To introduce our own graphical representation, we see first what characteristics of the π-calculus should be depicted graphically. An observation reveals that a name may play two roles in interactions as buffer and channel:

Buffer A name may work as a buffer in the sense that it can be read out, e.g., b in $\overline{a}\langle b\rangle$, or another name can overwrite it, e.g., x in $a(x)$.

Channel A name can work as a channel through which names can be exported, e.g., a in $\overline{a}\langle b\rangle$, or imported, e.g., a in $a(x)$.

The operations on them are accordingly categorised into two sorts, source and target arcs:

Source arc Source arcs represent the flow from buffer to channel. They are expressed as exporting action such as $\overline{a}\langle b\rangle$ in the π-calculus, where the name b is exported to the channel a.

Target arc Target arcs represent the flow from channel to buffer. They are expressed as importing action such as $a(x)$, where the name x will be replaced by another name imported through a.

Here is a clarification of the terminology of *channel*, *port*, and other notions related to them. As is shown in the figure 2.13, two buffers are connected via a channel. We call, thus, a in $(\overline{a}\langle b\rangle \mid a(x))$ *channel* as it connects b to x. Changing our perspective, when we observe the picture with our point of view in buffer, the left end of the channel can be seen as output *port* because a name can be exported there from the buffer to the channel. Similarly, the right end can be seen as input *port* because a name can be imported there from the channel to the buffer. Therefore, we may sometimes call a in $\overline{a}\langle b\rangle$ output port and input port if it is in $a(b)$.[9]

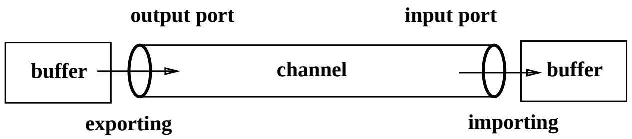


Figure 2.13. The relation between channel, port, and buffer

The above observation leads to a graph whose nodes are either buffer or channel and edges either source or target arcs. We should, however, pay attention to the ambiguity found in the graphic representation due to the two roles that can be played by a name at the same time if we depict the graph straightforwardly. If we depict relations between names simply as the graph shown in the figure 2.14, its meaning is ambiguous because it means either

(a) 'a' is a channel, exporting 'b' and importing it to 'c', or

(b) 'a' is a buffer, importing a name through 'b' and exporting it through 'c'.

As the example shows, we have to distinguish graphically these two aspects of names, which could be done in many ways. One way is to change its shape so that it consists of two different parts. A rectangle, for example, can be used for the purpose because it consists of pairs of *long* and *short* sides. When we depict names as rectangles, we can distinguish the two different roles, buffer and channel, by mapping them to different

[9] Alternatively, we can exchange *"output port"* and *"input port"* by taking channel as central. We are however not much concerned about the terminology since they will be rarely used in the following.

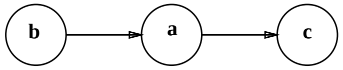


Figure 2.14. An ambiguous graph

length of sides (Figure 2.15). The presentation is, however, redundant because a role can be mapped to two different sides, e.g., buffer mapped to either top or bottom side.

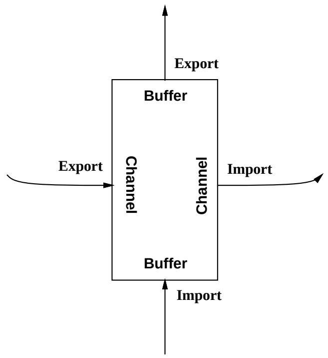


Figure 2.15. Name depicted as rectangle

Another way is to divide the node into two parts and paint them in different colours as is shown in the figure 2.16. In the figure, the buffer part of the node is painted *white* while the channel part *gray*. Arcs are also distinguished by different colours of heads. The heads of target arcs are painted *black* while source arcs *white*. The advantage of our colouring to shaping is that we can make it clear that the properties of being buffer or channel are attributed to 'nodes' while the properties of being source or target to 'arcs'. Choosing this way, we can present a system, $(\overline{a}\langle b\rangle \mid a(x))$, as is shown in the figure 2.17.

The representation could be improved if we leave nodes unshaded when they do not work as a channel.[10] If we adopt the convention, then the figure may be drawn as is shown in the figure 2.18, which clearly shows the names b and x are used only as

[10] The suggestion is due to Robin Cooper.

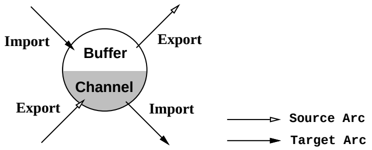


Figure 2.16. Name in two colours

buffer. We can further extend the discipline by painting different types of names with different colours or patterns.[11] We will however refrain from adopting the discipline here although we admit there is certainly a number of advantages with it. The discipline should be investigated more fully in another place.

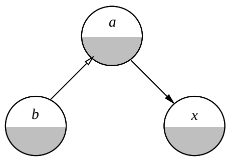


Figure 2.17. A representation of $(\overline{a}\langle b\rangle \ \mid a(x))$

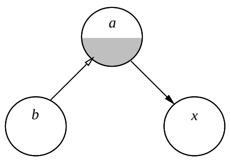


Figure 2.18. A modified version with unshaded nodes

[11] We will look into channel types in chapter 5.

2.2.3 π-nets

Before examining the underlying computation of the π-calculus in detail using the graphic notation, we look into our source of insights, π-net, proposed by Milner [Mil93a, Mil94] to present computation of the π-calculus graphically. The π-nets are basically a variation of rectangles presented above (Figure 2.15). We get a π-net node by allowing both sides of channel to import and export names and by allowing the buffer part to import only one name. For an aesthetic reason, it is depicted as a drop shown in Figure 2.19.

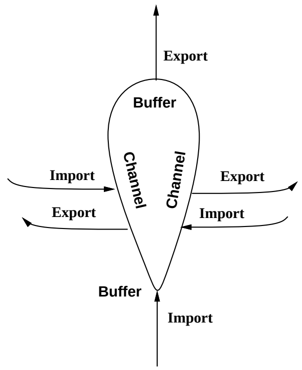


Figure 2.19. A π-net node

We have already discussed a disadvantage of shaping, which applies to the π-nets, too. Another point to be mentioned here is the restriction on importing to buffer that only one name can be imported. The intention of this restriction is to prohibits a name from being substituted by more than one names. In fact, such a case never occurs for the processes defined in the π-calculus. Observe, for example, the following case:

$$a(x).P \mid b(x).Q$$

Although the two occurrences of x seem to be identical and to be replaced by a name imported through either a or b non-deterministically, they are not and can be rewritten to the following:

$$a(x).P \mid b(y).Q$$

We do therefore not need to worry about the possibility of multiple-substitutions as long as we work with the π-calculus. We can however think of other languages that allow for such a case.[12] Due to the restriction, π-nets falls short of representing the case. Given our motivation to study varieties of communication, we prefer not impose the same restriction on our representation as the nets do. Our representation can be used to depict systems to which multiple-substitutions may occur.

2.3 Representing computation with dataflow graphs

Now that we have proposed our graphical representation, we elaborate it to depict the computation more fully.

2.3.1 Effects

Using the dataflow graphs we have presented above, a computation carried out by a system of processes, ($\overline{a}\langle b\rangle \mid a(x)$), can be depicted as is shown in the figure 2.20. We remove arcs after they are activated in order to show the computation denoted by them has been done. In the π-calculus notation, it can be expressed as: $\overline{a}\langle b\rangle.\mathbf{0} \mid a(x).\mathbf{0} \xrightarrow{\tau} \mathbf{0} \mid \mathbf{0}\{^b/_x\}$.

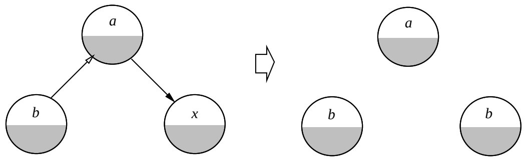


Figure 2.20. A presentation of $\overline{a}\langle b\rangle.\mathbf{0} \mid a(x).\mathbf{0} \xrightarrow{\tau} \mathbf{0} \mid \mathbf{0}\{^b/_x\}$

In the figure, the effect of the computation can be observed from the fact that two arcs have gone and the node named x is renamed to b. What is missing in the figure, however, is the substitution environment, $\{^b/_x\}$. As substitutions are the effects obtained as the result of computations, we should keep a record of them. To record such a substitution environment, we introduce a board. With the board, the computation can be depicted as is shown in the figure 2.21.

[12] The language $\mathcal{L}_0$ proposed in next chapter is one of the examples.

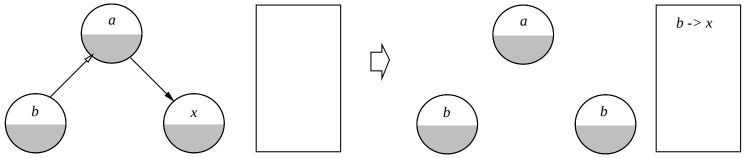


Figure 2.21. A representation with the board to record effects

2.3.2 Merging nodes

We will identify a node by its name stored in the buffer. Given this, when a name is substituted by some another name, it is always the case that two different nodes bear the same name as is observed from the above example (Figure 2.20). Once this happens, we will *merge* them into a single node. When nodes are merged, arcs pointed to or strung from them are merged, too. Figure 2.22 shows an example of merge. The figure depicts a computation carried out by the system, $(\nu\, b, c)(\overline{a}\langle b\rangle \mid a(c) \mid \overline{b}\langle e\rangle \mid c(d))$, whose transitions can be expressed as:

$$\begin{array}{llllll} & (\nu\, b, c) & (\overline{a}\langle b\rangle.\mathbf{0} & \mid a(c).\mathbf{0} & \mid \overline{b}\langle e\rangle.\mathbf{0} & \mid c(d).\mathbf{0}) \\ \xrightarrow{\tau} & (\nu\, b) & (\ \mathbf{0} & \mid \mathbf{0} & \mid \overline{b}\langle e\rangle.\mathbf{0} & \mid b(d).\mathbf{0}) \end{array}$$

In this system, the node c will be substituted by b after the interaction between $\overline{a}\langle b\rangle$ and $a(c)$, and the system will end up with two nodes named b. Then, they are going to be merged into the single b node, where their arcs are also merged.

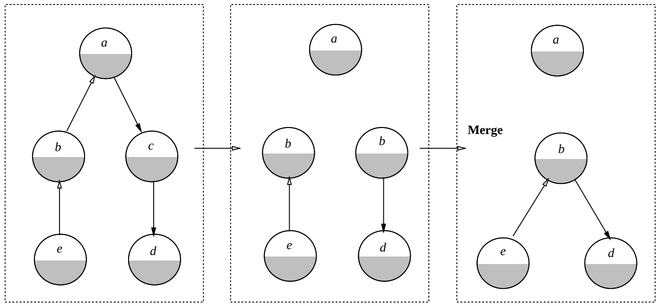


Figure 2.22. Merging nodes

2.3.3 Match

The match operator is used to control computation by suppressing the activation of particles. Recall in the system below, the particle $\overline{b}\langle e\rangle$ will never be performed because its match formula, $[x = 3]$, is not satisfied by its substitution environment, $\{^5/_x\}$.

$$\begin{array}{rllll} & \overline{c}\langle 5\rangle.\mathbf{0} & \mid c(x). & (\ [x=5]\ \overline{a}\langle d\rangle.\mathbf{0} & \mid [x=3]\ \overline{b}\langle e\rangle.\mathbf{0}) \\ \rightarrow & \mathbf{0} & \mid & [5=5]\ \overline{a}\langle d\rangle.\mathbf{0}\{^5/_x\} & \mid [x=3]\ \overline{b}\langle e\rangle.\mathbf{0}\{^5/_x\} \\ \xrightarrow{\overline{a}\langle d\rangle} & \mathbf{0} & \mid & \mathbf{0} & \mid [x=3]\ \overline{b}\langle e\rangle.\mathbf{0}\{^5/_x\} \end{array}$$

We put the match formulas just beside of arcs to show they control the execution of the particle (Figure 2.23). After the interaction between $\overline{c}\langle 5\rangle$ and $c(x)$, the substitution $\{^5/_x\}$ will be recorded to the board. Each arc with a match formula, e.g., $[x = 5]\overline{a}\langle d\rangle$, must inspect the board to check if its formula is satisfied with respect to the substitution environment. Since the formula $[x = 5]$ is satisfied by the environment, the particle $\overline{a}\langle d\rangle$ can then be performed, but not the other particle $\overline{b}\langle e\rangle$ because its formula, $[x = 3]$, is not satisfied by the substitution environment.

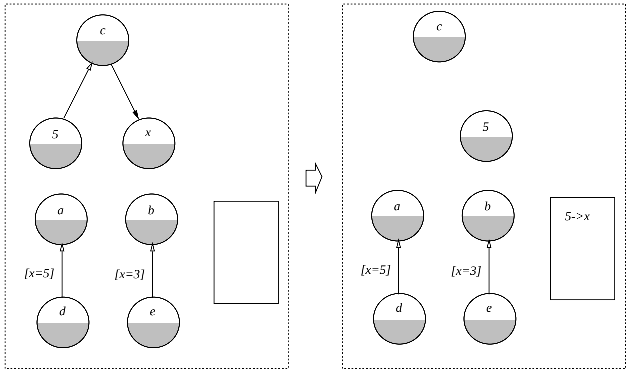


Figure 2.23. Representation of match with the effect board

2.3.4 Garbage collections

To base our presentation to a real machine as much as possible, we explicitly introduce garbage collection mechanism into our representation. The garbage that would soon be noticed is the nodes that are not connected by any arcs. In Figure 2.23, the nodes

named c and 5 are not connected by any arcs, therefore, can be eliminated (Figure 2.24 left).

Another possible garbage is the arcs with a match formula that will never be satisfied. Again in the same figure, the particle $\overline{b}\langle e\rangle$ will never be performed, therefore, can be eliminated (Figure 2.24 right). When the arc is eliminated, the two nodes linked by the arc, too, can be eliminated for the reason just mentioned.

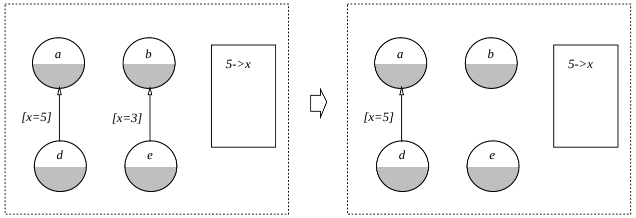


Figure 2.24. Garbage collection

2.3.5 Replication

While the meaning of replication is finitely many copies of processes, they are in practice not generated at once, but will be generated each time it is requested. That is, the transitions of a system of $!\overline{a}\langle b\rangle \mid a(x) \mid a(y)$ would be in practice as follows:

$$\begin{array}{rllll} & !\overline{a}\langle b\rangle & & \mid a(x).\mathbf{0} & \mid a(y).\mathbf{0} \\ \equiv & !\overline{a}\langle b\rangle & \mid \overline{a}\langle b\rangle.\mathbf{0} & \mid a(x).\mathbf{0} & \mid a(y).\mathbf{0} \\ \xrightarrow{\tau} & !\overline{a}\langle b\rangle & & \mid \mathbf{0}\{^b/_x\} & \mid a(y).\mathbf{0} \\ \equiv & !\overline{a}\langle b\rangle & \mid \overline{a}\langle b\rangle.\mathbf{0} & \mid \mathbf{0}\{^b/_x\} & \mid a(y).\mathbf{0} \\ \xrightarrow{\tau} & !\overline{a}\langle b\rangle & & \mid \mathbf{0}\{^b/_x\} & \mid \mathbf{0}\{^b/_y\} \end{array}$$

where $!\overline{a}\langle b\rangle$ replicates the particle $\overline{a}\langle b\rangle$ if there is a particle that can interact with it. In this sense, the particle with the replication operator is not in the *working memory*, but stored in a heap area from which a number of copies can be made upon request.

To visualise the separated area, we introduce a box to keep replicable particles. Given the machinery, the part of the above transitions can be depicted as is shown in Figure 2.25. In the figure, the gray arrows indicate that the particle $\overline{a}\langle b\rangle$ is spawned at the step from the box.

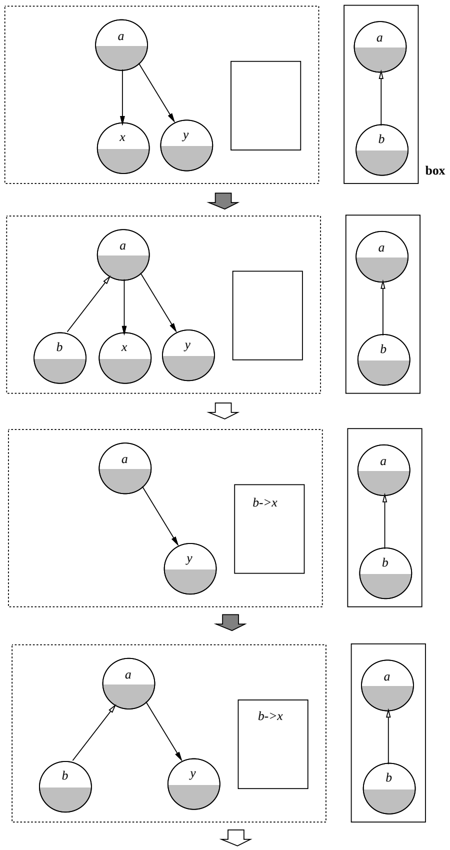


Figure 2.25. Replication with box

2.3.6 Free and bound names

The distinction between free and bound names may lead to wrong graphs without an additional step. Suppose there is a system composed of

$$a(x) \;|\, a(x) \;|\, \overline{a}\langle b\rangle$$

As the target of importing particles are bound, the two occurrences of x in the first and the second particle are not identical. Recall that the system is α-equivalent to the following one:

$$a(y) \;|\, a(z) \;|\, \overline{a}\langle b\rangle$$

If we do not care about this subtlety, the system may be depicted wrongly as is shown in the figure 2.26.

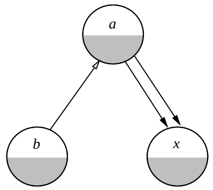


Figure 2.26. A wrong graph of $(a(x) \;|\, a(x) \;|\, \overline{a}\langle b\rangle)$

To avoid this kind of confusion, we number names with different natural numbers incrementally each time a binding occurs so that we can distinguish it from others bearing the same name. Thus, the above system is rewritten to:

$$a(x_1) \;|\, a(x_2) \;|\, \overline{a}\langle b\rangle$$

before being depicted graphically. With this step, the above figure can be drawn correctly as is shown in the figure 2.27. In the thesis, we may however suppress the subscripts where no confusion arises for readability.

2.3.7 Choice

The operator affects computation by allowing only one particle among candidates to be performed. For example, either $\overline{a}\langle x\rangle$.P or $\overline{b}\langle x\rangle$.Q, but not both, can be performed after the process receives 5 through c, replacing it for x, by executing $c(x)$.

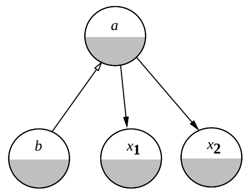


Figure 2.27. The correct graph of $(a(x)\ |\ a(x)\ |\ \overline{a}\langle b\rangle)$

$$\overline{c}\langle 5\rangle.\mathbf{0}\ |\ c(x).(\ \overline{a}\langle x\rangle.\mathrm{P} + \overline{b}\langle x\rangle.\mathrm{Q}\ )$$

To indicates the choice between $\overline{a}\langle x\rangle$.P and $\overline{b}\langle x\rangle$.Q, we may link those links with a particular arc (Figure 2.28). (The figure does not depict particles consisting of P and Q.) We ensure the other arc should be eliminated when either arc is activated by referring to choice arc.

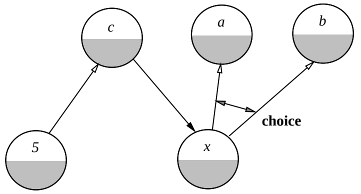


Figure 2.28. A choice arc between $\overline{a}\langle x\rangle$.P and $\overline{b}\langle x\rangle$.Q

The elimination operation can be slightly complicated to eliminate the succeeding particles in P and Q as well, but it can be done just by introducing another board to record sets of particles to be eliminated altogether.

2.4 Constraints on computation

We have so far confined ourselves to the description of the domain where computation occurs, but have not paid much attention to how it can be controlled except of *match* and *choice*. We now look into the control of computation in depth. When we discuss about *"control"*, we are concerned with how particles are executed, sequentially

or concurrently, and which particles can be executed under what condition. To control computation, the π-calculus, for example, provides us with three operators: *prefix*, *match*, and *choice*. After considering on the nature of control, we discuss what operations are appropriate to control computation.

2.4.1 Constraints

We start by observing what those dataflow graphs presented above denote. The graph shown in the figure 2.29, for example, can denote a number of processes defined in the π-calculus, among them are:

(a) $\overline{a}\langle b\rangle\ |\ a(c).\overline{d}\langle c\rangle\ |\ d(e)$

(b) $\overline{a}\langle b\rangle\ |\ a(c).\overline{d}\langle c\rangle.d(e)$

(c) $\overline{a}\langle b\rangle.a(c).\overline{d}\langle c\rangle\ |\ d(e)$

(d) $\overline{a}\langle b\rangle.a(c).\overline{d}\langle c\rangle.d(e)$

(e) $\overline{a}\langle b\rangle.d(e)\ |\ a(c).\overline{d}\langle c\rangle$

The reason why so many processes can be denoted is that the graph does not specify the order by which these particles are executed. Note that this is the necessary consequence of concurrency. Concurrent computation generalises sequential computation by not specifying the order of execution. An observation may reveal that the fifth process shown in the above is a specialisation of the first in that it ensures that $\overline{a}\langle b\rangle$ is executed before $d(e)$, while there is no mention to such an order in the first process.

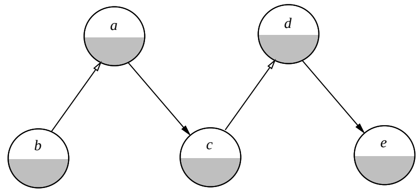


Figure 2.29. A dataflow graph

Given the observation, we take the denotation of graphs as the set of computation specified by them. We will, then, characterise the computation from two aspects. One is, as we observed just now, the *order* by which particles are executed, and the other is the *effects* that the interactions between particles bring about. The denotation of graphs are, therefore, the set of all possible execution orders of particles with their effects. It should be now clear how we can control computation. One way is, as provided in the π-calculus by the prefix operator, '**.**', to restrict the order of execution. By imposing conditions on execution by this way, we can slim down the set of possible computations by the graph. The other way is to restrict the effects to be brought about by the graph. The latter is not exploited in the π-calculus although it is the central idea in constraint logic programming and has been exploited in the modal mu-calculus [Sti92], too.

To sum up, processes can be defined with the set of particles and constraints on them:

$$\text{Processes} = \text{Particles} + \text{Constraints}$$

The particles form a dataflow graph as is shown above. In this section, we introduce two ways to define constraints corresponding to the two aspects of computation, order and effects.

2.4.2 Guarding

Our guarding operator generalises the prefix operator in the π-calculus. Our strategy to represent information on the order of execution is to separate it from the declaration of particles. Observe the following system of processes defined in the π-calculus:

$$\overline{a}\langle b\rangle.d(e) \mid a(c).\overline{d}\langle c\rangle$$

can be decomposed to a set of particles and constraints on them. As for particles, they consist of four particles, $\overline{a}\langle b\rangle$, $d(e)$, $a(c)$, and $\overline{d}\langle c\rangle$. As for constraints, two constraints are imposed, one of which is to execute $\overline{a}\langle b\rangle$ before $d(e)$ and the other is to execute $a(c)$ before $\overline{d}\langle c\rangle$, which we will express as $\overline{a}\langle b\rangle \prec d(e)$ and $a(c) \prec \overline{d}\langle c\rangle$, respectively. Given this notation, the above program may be rewritten as:[13]

$$[\,\overline{a}\langle b\rangle, d(e), a(c), \overline{d}\langle c\rangle \upharpoonright \overline{a}\langle b\rangle \prec d(e),\ a(c) \prec \overline{d}\langle c\rangle\,]$$

where $\upharpoonright$ separates the set of particles from constraints. The formulas to express the information on guarding can be elaborated by introducing boolean operators, but we discuss general principles for guarding before enriching the formulas.

The reason why we should discuss general principles is that we are concerned with Church-Rosser property of the calculus, which we obviously want to obtain. To understand hazardous points, observe first an ideal case, the graph shown in the figure 2.29. For the graph, we have two possible interactions, one of which is between $\overline{a}\langle b\rangle$ and $a(c)$, the other is between $\overline{d}\langle c\rangle$ and $d(e)$, where a and d work as channels. Of these two possible interactions, it does not matter which interaction occurs first because the result

[13] Processes are defined in this manner in the language $\mathcal{L}_0$ presented in next chapter.

is always the same as long as we are concerned with only substitution environments. For example, if the interaction between $\overline{d}\langle c\rangle$ and $d(e)$ occurs first (Figure 2.30), the resulting effects are the substitutions, $\{{}^{c}\!/_{e}, {}^{b}\!/_{c}\}$.

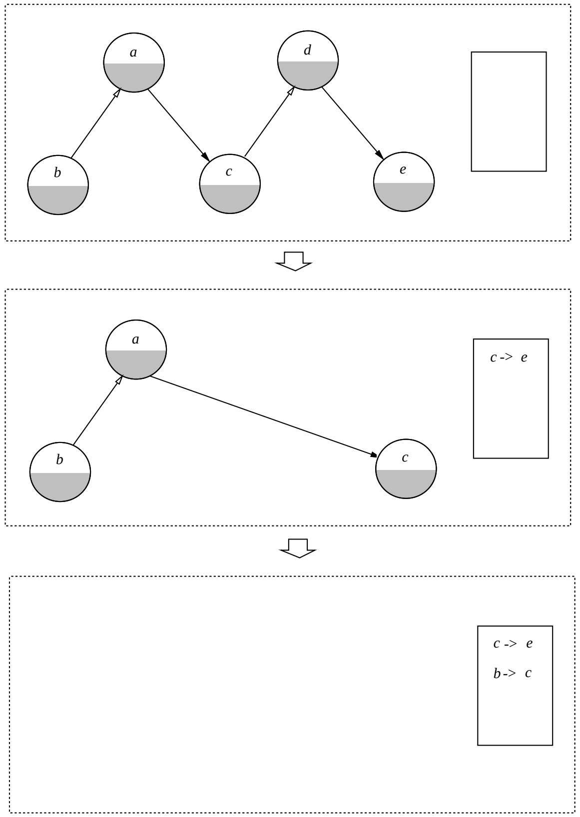


Figure 2.30. The case where the interaction between $\overline{d}\langle c\rangle$ and $d(e)$ occurs first

Meanwhile, if the interaction between $\overline{a}\langle b\rangle$ and $a(c)$ occurs first (Figure 2.31), the resulting effects are the substitutions, $\{{}^{b}\!/_{c}, {}^{b}\!/_{e}\}$.

Although these two sets of effects seem to be different at a glance, they are actually the same because both results support exactly the same constraints on substitutions, say b

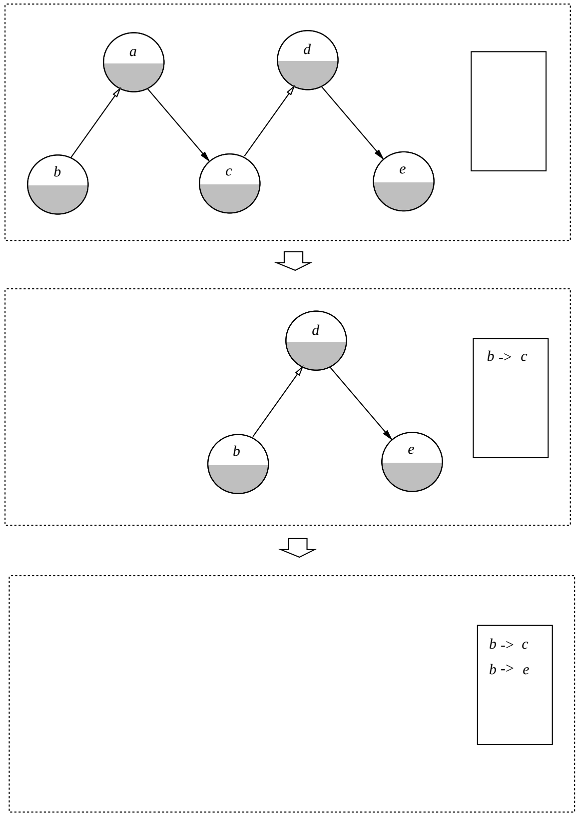


Figure 2.31. The case where the interaction between $\overline{a}\langle b\rangle$ and $a(c)$ occurs first

$= c = e$.

There is however a case where the order does matter. Suppose we add to the previous graph (Figure 2.30 or 2.31) the two arcs, $\overline{f}\langle a\rangle$ and $f(d)$. If we start the computation from bottom, we are simply getting another new effect, $\{{}^{a}\!/_{d}\}$, in addition to $\{{}^{b}\!/_{c}, {}^{c}\!/_{e}\}$, no matter how those bottom interactions are executed (Figure 2.32).

On the other hand, if we execute the interaction between $\overline{f}\langle a\rangle$ and $f(d)$ first, then another effect may be brought about, which was unavailable in the above case (Figure 2.33). By the first interaction, the node a substitutes d and the structure of the graph changes as is shown in the figure. Now there is a new possibility that c interacts with itself in addition to the previous cases. If c substitutes itself, then another possible effect is only $\{{}^{b}\!/_{e}\}$, and other effects $\{{}^{b}\!/_{c}, {}^{c}\!/_{e}\}$ will be lost.

The cause of this cumbersome situation is in the mobility enabled by the calculus. In the graph, the interaction between $\overline{a}\langle b\rangle$ and $a(c)$ occurs at a, and another between $\overline{d}\langle c\rangle$ and $d(e)$ occurs at d. We call these a and d the *location* of communication. What makes things complicated is these locations are mobile because they can communicate through f. As long as a node is mobile, allowing interactions at the location may eliminate some possible computations. The useful strategy to avoid such a situation is to allow communication to occur only at immobile locations. Such locations can be easily discovered in graphs. A node is immobile if there is no arrow pointed to or from in the buffer part. We will employ this principle when we define our language in next chapter. Our graphical representation is useful to detect immobile nodes because they always come up to top. In the figure, for example, only f node is immobile.

2.4.3 Temporal properties

Another way to restrict computation is to require it to satisfy particular formulas eventually with respect to substitution environments.[14] Observe an example, $(\overline{a}\langle b\rangle \mid \overline{a}\langle c\rangle \mid a(x) \mid a(y))$, which may end up with two cases of substitution environments (Figure 2.34). If the particle $\overline{a}\langle b\rangle$ interacts with $a(x)$, then the final substitution environment is $\{{}^{b}\!/_{x}, {}^{c}\!/_{y}\}$. On the other hand, if the particle interacts with $a(y)$, the environment results into $\{{}^{b}\!/_{y}, {}^{c}\!/_{x}\}$.

Assume we want only the first substitution environment to realise. If we resort to guarding, we may define the computation as:

$$[\, \overline{a}\langle b\rangle, \overline{a}\langle c\rangle, a(x), a(y) \restriction \overline{a}\langle b\rangle \prec \overline{a}\langle c\rangle,\ a(x) \prec a(y)\,]$$

[14] This sort of control is not included in the π-calculus. The original intention of looking into the sort of control was to build a basis rich enough to study concurrent constraint programming [Smo94a], but I found the subject going beyond the scope of the thesis. We will however in next chapter investigate to what extent the sort of control is computationally expensive.

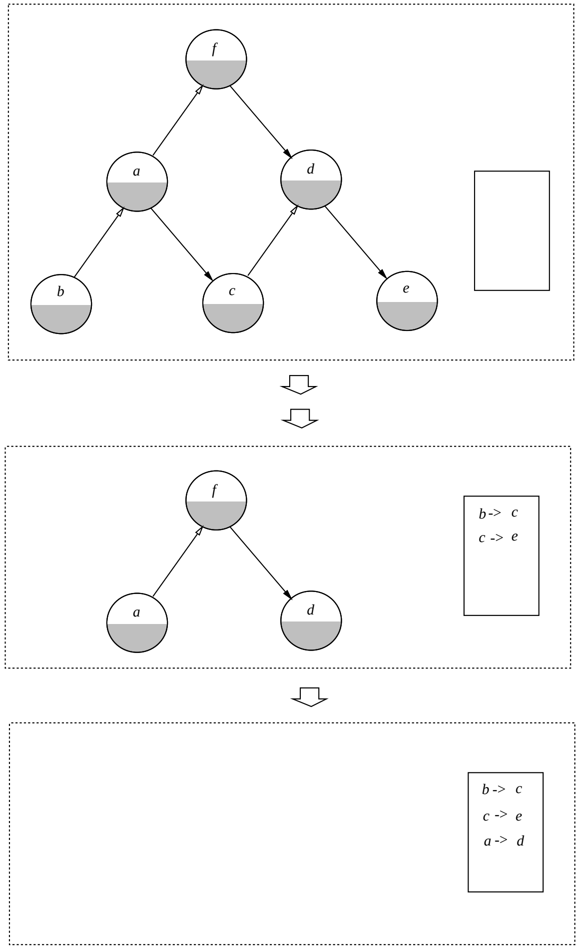


Figure 2.32. A mobile process interacting with from bottom

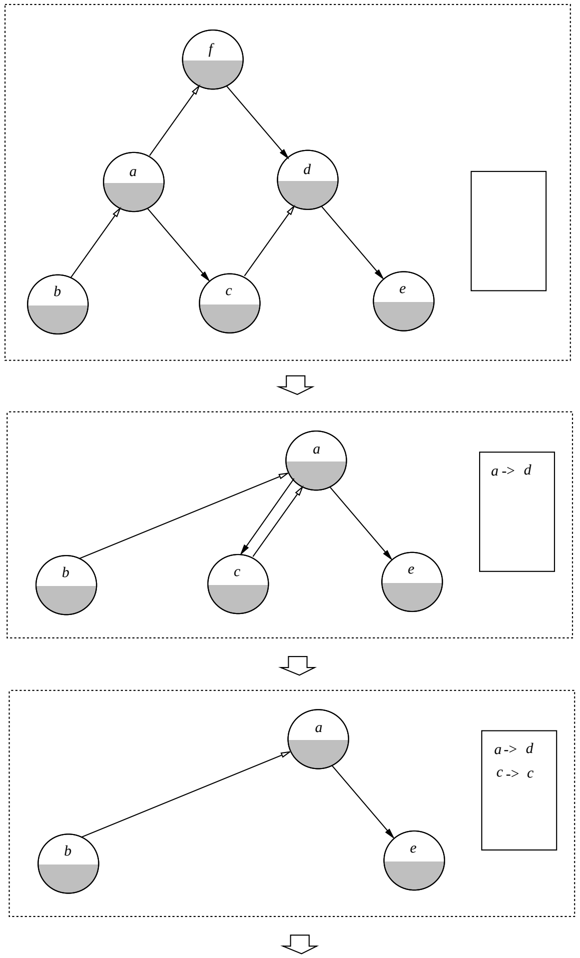


Figure 2.33. A mobile process interacting with from top

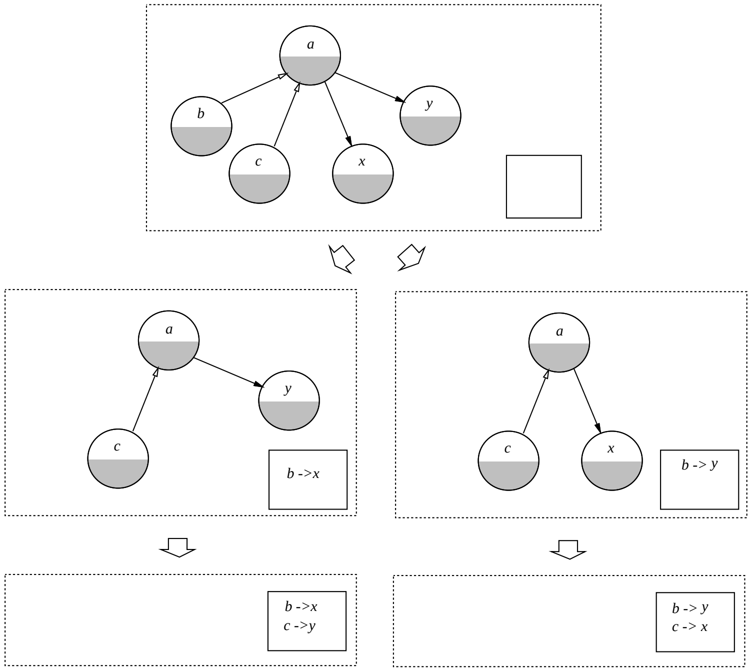


Figure 2.34. A dataflow graph with two possible substitution environments

or

$$[\,\overline{a}\langle b\rangle, \overline{a}\langle c\rangle, a(x), a(y) \upharpoonright \overline{a}\langle c\rangle \prec \overline{a}\langle b\rangle,\ a(y) \prec a(x)\,]$$

which leads to $\{^b/_x, ^c/_y\}$.

Although the encoding works, it is not neat to restrict execution order when what we really want is to get a particular substitution environment eventually. Also, while the encoding leads to the substitution environment, we may only want to ensure at least the first substitution, $\{^b/_x\}$, should be obtained. In that case, it does not make sense to restrict the execution order that way.

Our solution is to require the system to satisfy particular formulas eventually as the result of computation. If we want the system to satisfy at least a formula, $b = x$, we should add it to the system as a constraint as follows:

$$[\,\overline{a}\langle b\rangle, \overline{a}\langle c\rangle, a(x), a(y) \upharpoonright b = x\,]$$

Once this restriction is imposed on, only the first reduction is possible, but not the second is prohibited, as we want. We call the temporal property that "something good does eventually happen" *liveness*. The dual notion is *safety*, "nothing bad ever happens", which can be defined using negation, e.g., $b \neq y$.

Finally, we see how these operators, guarding ($\prec$) and temporal properties are related with match, e.g., $[x = y]\alpha$. Guarding and match share the same objective to restrict the execution of particles. The difference is that guarding suppresses an execution until another particle is executed, while match does the same job by inspecting substitution environments. The common property between match and temporal properties is that both inspect substitution environments, but they are different in that temporal properties restrict possible interactions while match restricts the execution of a single particle.

2.5 The reflexive π-calculus

Finally, we mention the reflexive π-calculus[Mil93a]. In the normal π-calculus, graphically speaking, it is not allowed to include *target-arc cycles* in graphs, while we allow them for our language. Target-arc cycles can be formed in two ways:

(a) A name exports its name to a node and imports it to itself (Figure 2.35), or

(b) A name imports a name from its own channel (Figure 2.36).

In the first figure (Figure 2.35), which can be expressed as $(\overline{b}\langle x\rangle \mid b(x))$, the name x is substituted by itself, keeping other names from substituting it. If we impose a restriction on processes such that a name can be substituted only by one name, this cycle effectively closes off itself from others. On the other hand, the second figure (Figure 2.36), which can be expressed as $\overline{x}\langle x\rangle$, shows a case where a buffer will be substituted by the name it is conveying.

One reason why these cycles are prohibited may be in its self-referential nature. But the reflexivity may be of some use and is admitted in the reflexive π-calculus. In the reflexive version, it is argued that the first sort of cycles may model the effect of the restriction operator, ν . However, the use of target-arc cycles is yet to be investigated.

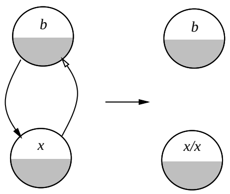


Figure 2.35. A target-arc cycle (1)

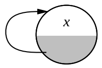


Figure 2.36. A target-arc cycle (2)

2.6 Summary

In this chapter, we looked into a model for concurrency through examination of the π-calculus. To make it easy to understand the computation, we have proposed a graphical representation based on π-nets. We have also seen that the computation can be controlled by guarding particles and requiring systems to satisfy temporal properties as the result of interactions.

Chapter 3

A logical specification of mobile processes

3.1 Introduction

The chapter presents a linear logical specification of mobile processes. The logic, on one hand, enables us to study communicating and mobile processes at certain abstract level and makes it possible, on the other hand, to relate the calculus of interaction with our interest, Situation Theory and Channel Theory, in logical and categorical setting.

We start this chapter by explaining the background of the project and general ideas. We then present a version of linear logic, combinatorial intuitionistic linear logic (§3.2), and a language $\mathcal{L}_0$ serving as a bridge between the π-calculus and the logic (§3.3). After explaining how dataflow graphs presented in the previous chapter can be utilised to predict possible interactions between processes (§3.4), we investigate the issue of non-determinism(§3.5). Finally, we discuss limitations of the logic and how the work can be related with other works (§3.6).

3.1.1 Background: Program Logic

One of the motivations to use logic to study processes is that it enables one to specify processes compositionally.[1] In such a logic, which is sometimes called Program Logic, programs or processes are taken to be models while formulas are regarded to express properties of processes. A theory is, therefore, often described in the form of $P \models \psi$, where P is a program and ψ a formula. With the logic, one should be able to identify a class of processes satisfying the same properties. It is one of the central problems in

[1] See, for example, Larsen [Lar90] for the discussion on specification formalisms.

computer science how to capture the sameness, which is termed *bisimulation* relation. The notion is important because it defines a model on which a program logic may be built. Different definitions lead to different program logics.

Hennessy-Milner logic (HML) [HM85] is a classical example of program logic, whose model is labelled transition systems (LTSs) [Mil89]. A LTS can be defined as triplet of *processes*, *actions*, and *transition relations* of processes upon actions, i.e., $(\mathcal{P}, Act, \xrightarrow{\alpha})$, where $\alpha \in Act$. They can be depicted as a tree whose nodes correspond to states of process and edges to actions. Suppose, for example, there is an agent A defined as $A =_{\text{def}} a.(b.\mathbf{0} + c.\mathbf{0})$.[2] It may be depicted as is shown in the figure 3.1.

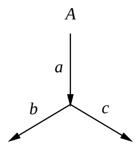


Figure 3.1. The tree representation of $a.(b.\mathbf{0} + c.\mathbf{0})$

A simple logic, $\mathcal{PL}$, over such trees may be defined as follows:

- $P \models \langle\alpha\rangle F$ if, for some P', $P \xrightarrow{\alpha} P'$ and $P' \models F$
- $P \models \neg F$ if it is not the case that $P \models F$
- $P \models \bigwedge_{i\in I} F_i$ if, for all $i \in I, P \models F_i$

We also define **true** as the empty conjunction $\bigwedge_{i\in 0} F_i$ and **false** as $\neg$**true**. With these definitions, the agent satisfies the following formula:

$$A \models \langle a\rangle(\langle b\rangle\mathbf{true} \wedge \langle c\rangle\mathbf{true})$$

Assume agents whose tree representations are identical are bisimular to each other, not bisimular otherwise. Let B be another agent defined as $B =_{\text{def}} a.b.\mathbf{0} + a.c.\mathbf{0}$. The agent may be depicted as is shown in the figure 3.2 and expressed in $\mathcal{PL}$ as:

$$B \models \langle a\rangle\langle b\rangle\mathbf{true} \wedge \langle a\rangle\langle c\rangle\mathbf{true}$$

Notice A and B are not bisimular although they can accept the same sequence of actions, $\overline{a}.\overline{b}$ or $\overline{a}.\overline{c}$. They can be distinguished by different tree representations or by different formulas specifying them. The difference between A and B characterised by HML or

[2] The example is taken from [Mil89]

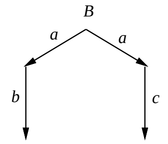


Figure 3.2. The tree representation of $a.b.\mathbf{0} + a.c.\mathbf{0}$

tree representation is the capability after accepting $\overline{a}$; A can accept $\overline{b}$ or $\overline{c}$ then, while B can only accept either one depending on the transition upon the interaction. In other words, the decision whether it will receive $\overline{b}$ or $\overline{c}$ is made when it receives $\overline{a}$, while such a decision is not made in A.

It is known that HML can characterise the bisimular relation sketched above [HM85]. It should be mentioned that a number of definitions have been proposed as for the equality between processes other than the one characterised by HML. For instance, one can define the relation based on traces, i.e., the set of sequences of possible actions. The processes, A and B, are then regarded to be equivalent if we are only concerned with traces. Comparing the bisimulation relation defined by HML with trace equivalence, one can say the former is more refined than the latter since the processes equivalent in HML are always so in terms of traces but its converse does not always hold. In this chapter, we will not go into the taxonomy of equivalence relations on processes since it is out of our scope.[3] It is enough for us to memorise the following three points:

(a) Processes may be depicted graphically.

(b) Logical formulas can specify the graphical representation.

(c) The graphic representation represents a model for the logic.

3.1.2 Linear Logic

Linear Logic is claimed to be able to specify concurrent computation at logical level [Gir87, Gir95]. The logic is said to be resource-bound as the rules for weakening and contraction, shown below in the form of classical logic, are dropped from structural rules, which allows a theorem prover to use a formula only once.

$$\begin{array}{ll} A \wedge B \Rightarrow A & \text{weakening} \\ A \Rightarrow A \wedge A & \text{contraction} \end{array}$$

[3] See Abramsky and Vickers [AV93] for a proposal, for example.

The restriction will in turn benefit us with the ability of expressing *current state*, which means that the implication relation can be regarded to model state changes such as chemical reaction. Metaphorically, we can express a chemical equation, $2H_2 + O_2 \rightarrow 2H_2O$, as a logical formula such as [4]

$$H_2 \otimes H_2 \otimes O_2 \multimap H_2O \otimes H_2O$$

where $\multimap$ is the linear implication. The connective, $\otimes$, which is sometimes called *tensor*, means intuitively that the composites are concurrently active.

We turn to linear logic to specify concurrent computation, which makes a shift from model-theoretic characterisation of processes, $P \models \phi$, to proof-theoretic specification, $\psi \vdash \phi$. The shift should be regarded as a refinement to program logic. As a stepping stone, we look into how petri nets, as an example of concurrent computation, may be specified using linear logic.

3.1.3 Linear logical specification of petri nets

Petri nets consists of *places* and *transitions*. The figure 3.3 shows the initial state of a petri net, whose places, depicted as circles, are a, b, c, d, e and f, and transitions are t and t' depicted as rectangles.[5] The dots in places indicate *tokens* for computation. The digit numbers on arrows from places to transitions indicate how many tokens may be consumed when the transition occurs. For example, the digit number 1 on the arrow from the place, a, to the transition, t, means one token stored in a will be taken away when t occurs. On the other hand, the digit numbers on arrows from transitions to places indicate how many tokens may be put in when the transition occurs. The digit number 3 between t and d, for example, means that three tokens will be put into d when t occurs.

The figure 3.4 shows the state of the petri net after the transitions, t and t', occured. Consumed tokens by these transitions are one token stored in a, three tokens in b, of which two are taken by t and one by t', and three tokens in c. The tokens put in are three tokens in d, three tokens in e, of which two are put by t and one by t', and four tokens in f.

To specify the transitions, t and t', in linear logic, we represent them as two special axioms such as:

$t : a \otimes b \otimes b \vdash d \otimes d \otimes d \otimes e \otimes e$, and

$t' : b \otimes c \otimes c \otimes c \vdash e \otimes f \otimes f \otimes f \otimes f$

[4] The example is taken from [Gir95].
[5] The example is taken from [MOM91].

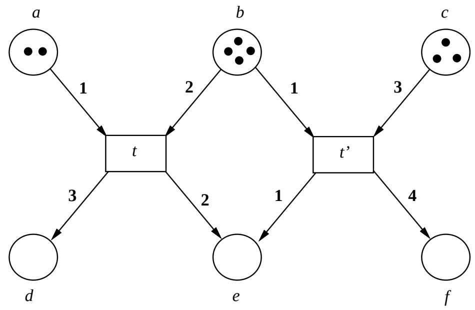


Figure 3.3. An initial state of a petri net

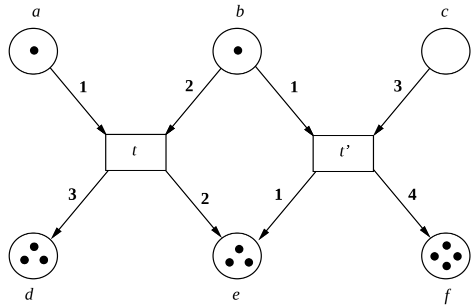


Figure 3.4. The state of the net after firing t and t'

where the number of alphabets, e.g., $b \otimes b$, corresponds to the number of tokens to be taken from or put into the place when the transition occurs. Suppose we adopt $\otimes$(tensor)-theory, a fragment of linear logic, which is defined with the axiom scheme and rules shown in the table 3.1.[6]

$$(id) \quad id_A : A \vdash A$$

$$(cut)\ \frac{\phi : A \vdash B \qquad \psi : B \vdash C}{\phi \circ \psi : A \vdash C} \qquad (\otimes)\ \frac{\phi : A \vdash B \qquad \psi : C \vdash D}{\phi \otimes \psi : A \otimes C \vdash B \otimes D}$$

Table 3.1. The axiom scheme and rules for tensor theory

With the axiom scheme and rules, the state change can be derived as below, where the repetition of alphabets is indicated with superscript, e.g., $b \otimes b = b^2$:

$$\frac{\dfrac{t : a \otimes b^2 \vdash d^3 \otimes e^2 \qquad t' : b \otimes c^3 \vdash e \otimes f^4}{t \otimes t' : a \otimes b^3 \otimes c^3 \vdash d^3 \otimes e^3 \otimes f^4}\ (\otimes) \qquad id : a \otimes b \vdash a \otimes b}{(t \otimes t') \otimes id : a^2 \otimes b^4 \otimes c^3 \vdash a \otimes b \otimes d^3 \otimes e^3 \otimes f^4}\ (\otimes)$$

Given this example, we may expect there would be some correspondence between linear logic and petri nets. The relationship between them has been studied on the basis of a category [MOM91] and an algebra [EW90, EW93]. Engberg and Winskel have shown that in fact a version of linear logic can be interpreted using petri nets and is complete on them [EW93]. The question we want to ask is whether we can do a similar thing to the π-calculus, that is, to capture the calculus in linear logic.

3.1.4 A petri net interpretation of CCS

To investigate the relationship between linear logic and the π-calculus, we first consider the relation between petri nets and CCS (Calculus of Communicating Systems) [Mil89], which can be thought of roughly as the π-calculus minus mobility [MPW91]. By relating CCS with petri nets, which can be a model for a version of linear logic, we can estimate what needs to be considered in specifying the π-calculus within linear logic.

Suppose there is a process, C, defined as [7]

[6] Strictly speaking, $\vdash$ should be written as $\vdash_\otimes$, but we write $\vdash$ for simplicity. In the following sections, we will actually adopt a combinatorial linear logic, not $\otimes$-theory, but it is enough at the moment to sketch basic ideas. Note also we write $\phi \circ \psi$ in defining *cut* rather than $\psi \circ \phi$ for convenience.

[7] The example is taken from [GM84] with a slight modification based on the material found in [BC94].

$$C =_{\text{def}} \ a.\mathbf{0} \mid \overline{a}.\mathbf{0}$$

The process can be translated to a petri net shown in the figure 3.5, where t means a silent action. In the net, a transition may occur either by firing a and $\overline{a}$ independently or by firing t, but not by firing both, because the tokens in p and q will be consumed when t is fired, which prohibits a and $\overline{a}$ from firing and vice versa. In either case, however, the final state is the same (Figure 3.6).

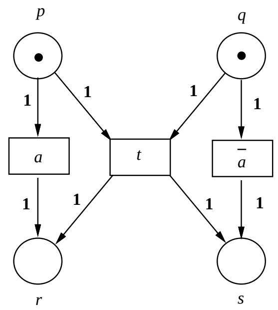


Figure 3.5. A petri net translation of $a.\mathbf{0} \mid \overline{a}.\mathbf{0}$

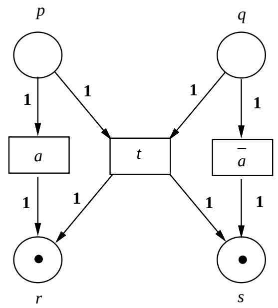


Figure 3.6. The final state of the petri net

Analogous to the linear logical specification of petri nets, one may think of specifying the CCS programs by adding the following axioms to $\otimes$-theory. The state change can be inferred either by t alone or by applying the $\otimes$-rule to a and $\overline{a}$, i.e., $a \otimes \overline{a} : p \otimes q \vdash r \otimes s$.

(a) $a : p \vdash r$

(b) $\overline{a} : q \vdash s$

(c) $t : p \otimes q \vdash r \otimes s$

We can now ask how the states, p, q, r, and s, can be characterised. Recall that the precondition and postcondition are defined in terms of tokens consumed or produced upon transitions for the case of petri-nets. By applying the idea to CCS, we can see the state, p, can for example be characterised with the resource to perform the action, a. The state, q, then can analogously be characterised as $\overline{a}$. We replace u and v for a and $\overline{a}$ as term and characterise the states, r, s, and $r \otimes s$ as $\mathbf{0}$, [8] the state where no resource is available. After the modification, the figure 3.5 and 3.6 are redrawn to 3.7 and 3.8, respectively.[9]

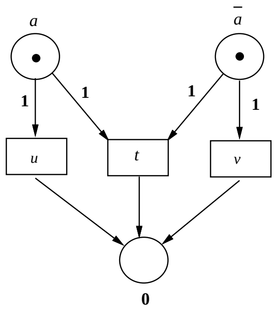


Figure 3.7. A modified petri net of $a.\mathbf{0} \mid \overline{a}.\mathbf{0}$

The axioms too are redefined as below. One may notice the first two of the axioms correspond to the process definition in the π-calculus that $a.\mathbf{0}$ and $\overline{a}.\mathbf{0}$, respectively. We conceive of the π-calculus definition of processes as stating state changes.

(a) $u : a \vdash \mathbf{0}$

(b) $v : \overline{a} \vdash \mathbf{0}$

(c) $t : a \otimes \overline{a} \vdash \mathbf{0}$

[8] We will later replace $\mathbf{1}$ for $\mathbf{0}$.

[9] Strictly speaking, it is odd to depict $\mathbf{0}$ as if it were a place, but it is convenient to indicate the transition.

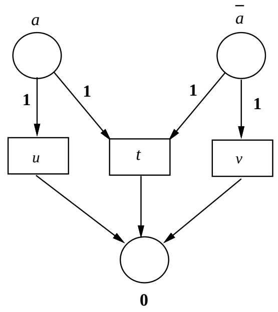


Figure 3.8. The final state of the petri net

We have seen it is possible to translate CCS processes to petri nets as long as they do not exchange messages. But how can we translate them if they exchange messages? Consider for example the following program C', where x will substitute y upon the interaction through a:

$$C' =_{\text{def}} a(y).\mathbf{0} \mid \overline{a}\langle x\rangle.\mathbf{0}$$

How can the effect that y is substituted by x be expressed? Petri nets cannot express the extra information on substitutions, but its linear logical specification can, by adding an extra formula to the third axiom to define the effect:

(a) $u : a(y) \vdash \mathbf{0}$

(b) $v : \overline{a}\langle x\rangle \vdash \mathbf{0}$

(c) $t : a(y) \otimes \overline{a}\langle x\rangle \vdash \mathbf{0} \otimes\ !(x = y)$

Here, the formula, $!(x = y)$, added to t expresses the effect upon the interaction between $a(y)$ and $\overline{a}\langle x\rangle$. The of-course operator, !, ensures the information can be provided for as many times as it is requested. This is because the information on substitution should be available forever. Another point is that we express substitutions as the equality between names, e.g., $x = y$.

The encoding of substitutions as equations makes it possible to define MATCH. Suppose in the program C' the execution of $\overline{a}\langle x\rangle$ was restricted by a match formula, $b = c$:

$$C'' =_{\text{def}} \ a(y).\mathbf{0} \mid [b=c]\overline{a}\langle x\rangle.\mathbf{0}$$

The restriction can be expressed by adding the extra condition to v and t:

(a) $u : a(y) \vdash \mathbf{0}$

(b) $v : \overline{a}\langle x\rangle \otimes\ !(b=c) \vdash \mathbf{0}$

(c) $t : a(y) \otimes \overline{a}\langle x\rangle \otimes\ !(b=c) \vdash \mathbf{0} \otimes\ !(x=y)$

By adding to them the formula, $!(b=c)$, we can ensure that the axioms v and t are applied to only when the condition is satisfied, which may be thought of as simulating the effect by MATCH.

Now that it becomes clear that the petri net translation falls short of expressing the cases where messages are exchanged, we turn back to our graphical representation, dataflow graphs, and consider how it can be related to linear logic. Let us see first how the program C' may be depicted in the graph (Figure 3.9).

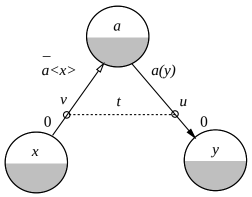


Figure 3.9. A dataflow graph translation of $a(y).\mathbf{0} \mid \overline{a}\langle x\rangle.\mathbf{0}$

In the figure, v and u on the arrows indicate their corresponding actions. t on the line connecting v and u indicates the interaction between them. As depicted in the figure, we put the states before action in upper and those after action in lower part of arrows. This is because we execute computation from upper parts depicted in the figure to lower parts. We will fill in the lower part of arrows with resources made available by executing the action, which are typically resources for actions immediately guarded by it.[10] The translation from dataflow graphs to linear logic is straightforward. We start by observing each arrow, u and v, and translate them into $u : a(y) \vdash \mathbf{0}$ and $v : \overline{a}\langle x\rangle \vdash \mathbf{0}$, respectively. t is first translated to $t : a(y) \otimes \overline{a}\langle x\rangle \vdash \mathbf{0}$, and then $!(x=y)$ is added to its consequence by inspecting the nodes connected by t.

[10] More detailed explanation can be found in §3.6.4.

3.1.5 A linear logical specification of the π-calculus

The final step is to express *mobility* in linear logic. How can for example the following program be specified, where the channel b is passed to the first process from the second?

$$D =_{\text{def}} \ a(y).y(z).\mathbf{0} \mid \overline{a}\langle b\rangle.\overline{b}\langle x\rangle.\mathbf{0}$$

In our graphical representation, the program may be depicted and be decorated as is shown in the figure 3.10. The arrow v_2 is decorated with $\overline{b}\langle x\rangle$ and $\mathbf{0}$, and another arrow u_2 with $y(z)$ and $\mathbf{0}$. $\overline{b}\langle x\rangle$ and $y(z)$ appear in the lower part of the arrows above them, too, which means the resources to execute u_2 and v_2 are provided for by executing u_1 and v_1, respectively. t_1 is same as t presented previously. We have to be cautious in specifying mobility, however. If we specify t_2, the possible interaction between v_2 and u_2, in the same way as t_1, it might be expressed as:

$$t_2 : y(z) \otimes \overline{b}\langle x\rangle \vdash \mathbf{0} \otimes\ !(x=z)$$

But the specification is incomplete as it allows the interaction to occur even when there is no interaction between v_1 and u_1. Since the interaction between v_2 and u_2 becomes possible only when b substitutes y, we should express it by adding to the antecedent the extra condition $(b=y)$:

$$t_2 : y(z) \otimes \overline{b}\langle x\rangle \otimes\ !(b=y) \vdash \mathbf{0} \otimes\ !(x=z)$$

The processes are therefore specified as follows:

(a) $u_1 : a(y) \vdash y(z)$

(b) $u_2 : y(z) \vdash \mathbf{0}$

(c) $v_1 : \overline{a}\langle b\rangle \vdash \overline{b}\langle x\rangle$

(d) $v_2 : \overline{b}\langle x\rangle \vdash \mathbf{0}$

(e) $t_1 : a(y) \otimes \overline{a}\langle b\rangle \vdash y(z) \otimes \overline{b}\langle x\rangle \otimes\ !(b=y)$

(f) $t_2 : y(z) \otimes \overline{b}\langle x\rangle \otimes\ !(b=y) \vdash \mathbf{0} \otimes\ !(x=z)$

Finally, we replace $\mathbf{1}$ for $\mathbf{0}$ and assume an axiom $\mathbf{1} \otimes \mathbf{1} \vdash \mathbf{1}$, which holds in the logic. With these axioms, we can infer the transition from the initial state, where the actions, $a(y)$ and $\overline{a}\langle b\rangle$, are possible, to the final state, where no action is possible with the substitution environment, $x=z$ and $b=y$ as follows:

$$\dfrac{t_1 : a(y) \otimes \overline{a}\langle b\rangle \vdash y(z) \otimes \overline{b}\langle x\rangle \otimes\ !(b=y) \qquad \dfrac{\dfrac{t_2 : y(z) \otimes \overline{b}\langle x\rangle \otimes\ !(b=y) \vdash \mathbf{1} \otimes\ !(x=z) \qquad \dfrac{id : (b=y) \vdash (b=y)}{!id : !(b=y) \vdash !(b=y)}\,!}{t_2 \otimes !id : y(z) \otimes \overline{b}\langle x\rangle \otimes\ !(b=y) \otimes\ !(b=y) \vdash \mathbf{1} \otimes\ !(x=z) \otimes\ !(b=y)}\,\otimes}{t_2 \otimes !id : y(z) \otimes \overline{b}\langle x\rangle \otimes\ !(b=y) \vdash \mathbf{1} \otimes\ !(x=z) \otimes\ !(b=y)}\,con}{t_1 \circ (t_2 \otimes !id) : a(y) \otimes \overline{a}\langle b\rangle \vdash \mathbf{1} \otimes\ !(x=z) \otimes\ !(b=y)}\,cut$$

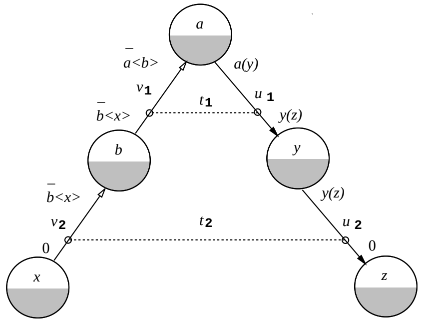


Figure 3.10. A dataflow graph translation of $a(y).y(z).\mathbf{0} \mid \overline{a}\langle b\rangle.\overline{b}\langle x\rangle.\mathbf{0}$

In this derivation, we assume a contraction rule shown below, which will be proved valid in our logic shortly (§3.2.2). The term can also be reduced to $t_1 \circ t_2$ using the equations given in the table 3.7.

$$\frac{A \otimes \ (!B \otimes \ !B) \vdash C}{A \otimes \ !B \vdash C}\ con$$

3.1.6 The strategy

In this chapter, we show how mobile processes can be specified using linear logic. To explain the translation, we go through the following steps:

$$\begin{array}{c}\text{processes defined in the } \pi\text{-calculus}\\ \Downarrow\\ \text{expressions in } \mathcal{L}_0\\ \Downarrow\\ \text{decorated dataflow graphs}\\ \Downarrow\\ \text{linear logic formulas}\end{array}$$

The language $\mathcal{L}_0$ serves as an intermediate language between the π-calculus and decorated dataflow graphs, and can simulate a fragment of the π-calculus without replication, !, and choice, +. The reason why we employ the language is that the π-calculus is too restrictive in terms of prefixing to encode some of the situation-theoretic objects in next chapter. Since the restriction is not essential, the language can be seen as a generalisation of the calculus.[11] We could therefore have started the previous chapter by

[11]The improvement has been proposed by Milner [Mil92b].

presenting the language, skipping the explanation about the π-calculus. But we opted for starting by examining the calculus because it makes it easier to understand the ideas of communication and concurrency. One should however not be misguided to believe that our aim in this chapter is limited to define processes in the π-calculus with linear logic.

3.2 A Combinatorial Linear Logic

We turn to a combinatorial intuitionistic linear logic (C-**ILL**), which may be regarded as a system for deriving sequent of the form $A \Rightarrow B$. A rule for sequential composition is defined for example as follows:

$$\frac{a : A \Rightarrow B \quad b : B \Rightarrow C}{a \circ b : A \Rightarrow C}$$

The choice of combinatorial linear logic rather than usual intuitionistic linear logic defined as natural deduction systems (N-**ILL**) is motivated by our interest; Its connection with category theory help us to relate our system with Situation and Channel theories. Note that we loose nothing by turning to combinatorial linear logic. It is known that N-**ILL** and C-**ILL** are equivalent in the sense that if $\phi : A \Rightarrow B$ is derivable in C-**ILL**, there is a deduction of $A \vdash B$ in N-**ILL**, and if $A_1, \ldots, A_n \vdash B$ in N-**ILL**, there is a term ϕ such that $\phi : A_1 \otimes \cdots \otimes A_n \Rightarrow B$ is derivable in C-**ILL** ([Laf88, Tro92]).

3.2.1 Definition of Combinatorial Intuitionistic Linear Logic

The Table 3.2 lists the axioms and rules for Combinatorial Intuitionistic Linear Logic without exponentials, C-$\mathbf{ILL}_0$.[12] As is mentioned above, we write $\phi \circ \psi$, not $\psi \circ \phi$, when two sequences ϕ and ψ are sequentially composed. $\mathbf{1}$ means a *unit*, and $\top$ and $\mathbf{0}$ *top* and *bottom*, respectively.[13] In the following, we may drop for simplicity the subscripts, e.g. A, B, and C in id_A and $cur_{A,B,C}$.

The intuitive meaning of connectives can be explained as follows:

- tensor ($\otimes$): $A \otimes B$ means both A and B are available or concurrently active.
- with (&) : $A \,\&\, B$ means only one of them is available, but the choice is up to the theorem prover.
- plus ($\oplus$) : $A \oplus B$ means either one can be chosen, but the theorem prover has no right in the decision.

[12]The source of the section on combinatorial linear logic is [Tro92], Chapter 9.
[13]The relation between $\mathbf{1}$, $\top$, and $\mathbf{0}$ is confusing because not ($\mathbf{1}$, $\mathbf{0}$) but ($\top$, $\mathbf{0}$) are dual. The symbol, $\perp$, is reserved for multiplicative disjunction, which is excluded from combinatorial intuitionistic linear logic.

Sequential composition:

$$\frac{\phi : A \Rightarrow B \quad \psi : B \Rightarrow C}{\phi \circ \psi : A \Rightarrow C} \qquad id_A : A \Rightarrow A$$

Parallel composition:

$$\frac{\phi : A \Rightarrow B \quad \psi : C \Rightarrow D}{\phi \otimes \psi : A \otimes C \Rightarrow B \otimes D} \qquad 1 : \mathbf{1} \Rightarrow \mathbf{1}$$

Adjointness of $\multimap$ and $\otimes$:

$$\frac{\phi : A \otimes B \Rightarrow C}{cur_{A,B,C(\phi)} : A \Rightarrow B \multimap C} \qquad ev_{A,B} : (A \multimap B) \otimes A \Rightarrow B$$

Symmetry, associativity, and unit:

$$\gamma_{A,B} : A \otimes B \Rightarrow B \otimes A$$

$$\alpha_{A,B,C} : A \otimes (B \otimes C) \Rightarrow (A \otimes B) \otimes C \qquad \alpha^{-1}_{A,B,C} : (A \otimes B) \otimes C \Rightarrow A \otimes (B \otimes C)$$

$$\lambda_A : \mathbf{1} \otimes A \Rightarrow A \qquad \lambda^{-1}_A : A \Rightarrow \mathbf{1} \otimes A$$

Products:

$$\frac{\phi : A \Rightarrow B \quad \psi : A \Rightarrow C}{\langle \phi, \psi \rangle : A \Rightarrow B \& C} \qquad \pi_{A,B,i} : A_0 \& A_1 \Rightarrow A_i \ (i \in \{0, 1\})$$

$$\top_A : A \Rightarrow \top$$

Coproducts:

$$\frac{\phi : B \Rightarrow A \quad \psi : C \Rightarrow A}{[\,\phi, \psi\,] : B \oplus C \Rightarrow A} \qquad \kappa_{A,B,i} : A_i \Rightarrow A_0 \oplus A_1 \ (i \in \{0, 1\})$$

$$\mathbf{0}_A : \mathbf{0} \Rightarrow A$$

Table 3.2. Axioms and rules for Combinatorial Intuitionistic Linear Logic, C-$\mathbf{ILL}_0$

The difference between & and $\oplus$ lies in the different sources of non-determinism: outer and inner non-determinisms, a notion investigated in computer science. Note the 'with' operator, &, should not be regarded as disjunction although the operator certainly exhibits the feature. Observe the following example to understand its intuitive meaning, where you can only buy one bottle of red or white wine for £2.99: [14]

Wine of the month	
Red	
&	Only for £2.99!
White	

3.2.2 Axioms and rules for the exponential

The combinatory intuitionistic linear logic with the exponential, ! (of-course), abbreviated as C-$\mathbf{ILL}_e$, is defined by adding to C-$\mathbf{ILL}_0$ (Table 3.2) the axioms and rules shown in the table 3.3.

Exponential:

$$\frac{\phi : A \Rightarrow B}{!\phi : !A \Rightarrow !B} \qquad \mathbf{s}_A : !A \Rightarrow !!A \qquad \mathbf{r}_A : !A \Rightarrow A \qquad \mathbf{t} : !\top \Rightarrow \mathbf{1}$$

$$\mathbf{p}_{A,B} : !(A \& B) \Rightarrow !A \otimes !B \qquad \mathbf{p}^{-1}_{A,B} : !A \otimes !B \Rightarrow !(A \& B)$$

Table 3.3. Axioms and rules for the exponential

Once these axioms and rules are added to C-$\mathbf{ILL}_0$, *thinning* and *contraction* are recovered as is shown in the table 3.4 and 3.5, respectively.

Thinning:

$$\frac{B \Rightarrow C}{!A \otimes \ B \Rightarrow C}$$

Derivation:

$$\dfrac{\dfrac{\dfrac{A \Rightarrow \top}{!A \Rightarrow !\top}\,! \quad B \Rightarrow B}{!A \otimes B \Rightarrow !\top \otimes B}\,\otimes \quad \dfrac{\dfrac{!\top \Rightarrow \mathbf{1} \quad B \Rightarrow B}{!\top \otimes B \Rightarrow \mathbf{1} \otimes B}\,\otimes \quad \dfrac{\mathbf{1} \otimes B \Rightarrow B \quad B \Rightarrow C}{\mathbf{1} \otimes B \Rightarrow C}\,\circ}{!\top \otimes B \Rightarrow C}\,\circ}{!A \otimes \ B \Rightarrow C}\,\circ$$

Table 3.4. The rule for thinning and its derivation

[14] In the original, & was +, which I found at a shop, *Margiotta* in Marchmont, Edinburgh.

Contraction:

$$\frac{A\otimes\ (!C\otimes\ !C)\Rightarrow D}{A\otimes\ !C\Rightarrow D}$$

Derivation:

$$\dfrac{\dfrac{A\Rightarrow A \quad !C\Rightarrow !C\otimes\ !C}{A\otimes\ !C\Rightarrow A\otimes\ (!C\otimes\ !C)}\otimes \qquad A\otimes (!C\otimes\ !C)\Rightarrow D}{A\otimes\ !C\Rightarrow D}\circ$$

The derivation of $!C\Rightarrow !C\otimes\ !C$:

$$\dfrac{\dfrac{!C\Rightarrow !!C \qquad \dfrac{\dfrac{\dfrac{!C\Rightarrow C \qquad \dfrac{C\Rightarrow C \quad C\Rightarrow C}{C\Rightarrow C\&\ C}\&}{!C\Rightarrow C\&\ C}\circ}{!!C\Rightarrow !(C\&\ C)}\,!}{!C\Rightarrow !(C\&\ C)}\circ \qquad !(C\&\ C)\Rightarrow !C\otimes\ !C}{!C\Rightarrow !C\otimes\ !C}\circ$$

Table 3.5. The rule for contraction and its derivation

3.2.3 The axioms for substitutions

We need special axioms to incorporate substitutions into the logic. They should, for example, allow for the following inference, where every occurrence of d in the consequence is replaced by c when the information contained in $!(c=d)$ is consumed.

$$\frac{A\Rightarrow\ !(c=d)\otimes\ !(d=e)}{A\Rightarrow\ !(c=e)}$$

For this purpose, we extend the logic by introducing a predicate expressing equality or identity between names. Let the symbol '=' be the predicate, which takes two arguments. We write 'c=e' for '=(c,e)' as usual and call the extended logic C-$\mathbf{ILL}_e^{=}$ because it is C-$\mathbf{ILL}_e$ plus the equality predicate.

The axioms for '=' are defined as is shown in the table 3.6.[15] The first rule $\mathbf{eq}_1$ defines the relation as reflexive and states the formula, $!(t=t)$, holds without any assumptions. The second rule defines the relation as symmetric. The third one defines the rule for substitutions. $[t/x]$ in $P[t/x]$ means the occurrence of t standing for x in P, and $P[s/x]$ means to replace s for every x. In the table, the alphabets s or t, connected with '=', must be a name, not a proposition.

Given the set of rules, we can check that '=' is transitive, i.e., $!(s=t)\otimes\ !(t=u)\Rightarrow\ !(s=u)$, directly from $\mathbf{eq}_3$ because $\mathbf{eq}_3 : !(s=t)\otimes\ !(t=u)[t/x]\Rightarrow\ !(s=u)[s/x]$. Note we could obtain the same result even if the two formulas in the antecedent were swapped,

[15] More succinctly, one can define the axioms without ! with an additional axiom, $\mathbf{eq}_1^- : (t=t)\Rightarrow \mathbf{1}$.

Equality:

$\mathbf{eq}_1 : \mathbf{1}\Rightarrow\ !(t=t)$

$\mathbf{eq}_2 : !(s=t)\Rightarrow\ !(t=s)$

$\mathbf{eq}_3 : !(s=t)\otimes P[t/x]\Rightarrow P[s/x],$

Table 3.6. The axioms for equality

i.e., $!(t=u)\otimes\ !(s=t)\Rightarrow\ !(s=u)$, because the antecedent can be transformed by $\mathbf{eq}_2$ to $!(u=t)\otimes\ !(s=t)$, and then $\mathbf{eq}_3$ can be applied to, i.e., $!(u=t)\otimes\ !(s=t)[t/x]\Rightarrow\ !(s=u)[u/x]$.

3.3 A molecular language $\mathcal{L}_0$

The language $\mathcal{L}_0$ is a syntactic variant of the π-calculus without choice, +, and replication, !, but less restrictive in the use of prefixing. It is equipped with the restriction, ν, but the effects are treated differently. Translating processes in the π-calculus into the language does not alter the semantics within the limitation.

3.3.1 Parallel composition

We assume by default particles are concurrently active and do not introduce a particular symbol to the language for parallel composition. For instance, a definition of process in the π-calculus:

$$\overline{a}\langle b\rangle \ \mid a(b)$$

can be expressed as:

$$[\,\overline{a}\langle b\rangle, a(b)\,]$$

In the following, we call the unit indicated by [and] *molecule* in that it is distinguished by the boundary.[16] We allow molecules to be nested in other molecules. For instance, the following expression is legitimate in our language.

$$[\,[\,\overline{a}\langle b\rangle, a(b)\,], [\,\overline{c}\langle d\rangle, c(e)\,], \overline{g}\langle h\rangle\,]$$

[16] The metaphor of molecules is proposed by Berry and Boudol [BB92]. The notation is adopted from [Mil93a].

3.3.2 Guarding

The specification of the guarding is separated from the declaration of particles, e.g., a process,

$$\overline{a}\langle b\rangle.\overline{c}\langle d\rangle.\mathbf{0}$$

is translated to:

$$[\, \overline{a}\langle b\rangle, \overline{c}\langle d\rangle \upharpoonright \overline{a}\langle b\rangle \prec \overline{c}\langle d\rangle \,]$$

where "$\overline{a}\langle b\rangle$, $\overline{c}\langle d\rangle$" preceding $\upharpoonright$ declares particles comprising the program, and "$\overline{a}\langle b\rangle \prec \overline{c}\langle d\rangle$" following it expresses how these particles are guarded, i.e., $\overline{a}\langle b\rangle$ guards $\overline{c}\langle d\rangle$.

The tree structure of guarding relations

We impose a restriction in constraining the execution of particles with guarding. We ensure that a particle should be guarded by at most one particle. That is,

$$[\, a(x), b(y), c(z) \upharpoonright a(x) \prec b(y),\ a(x) \prec c(z) \,]$$

is admissible[17] since both particles, $b(y)$ and $c(z)$, are guarded by only one particle, $a(x)$, but the following expression:

$$[\, a(x), b(y), c(z) \upharpoonright a(x) \prec b(y),\ c(z) \prec b(y) \,]$$

is not since $b(y)$ is guarded by two particles, $a(x)$ and $c(z)$. More graphically, the guarding relations should form a tree as is shown in the figure 3.11, not a graph (Figure 3.12). [18]

In the π-calculus, the syntax is defined so that it does not allow for the construction forming a graph. Recall that a program such as:

$$a(x).(b(y).\mathbf{0} \mid c(z).\mathbf{0})$$

is legitimate, but the below is not.

$$(a(x).\mathbf{0} \mid c(z).\mathbf{0}).b(y).\mathbf{0}$$

[17] Hereinafter, we may omit $\mathbf{1}$ and its relating guarding information for simplicity. The expression should have been $[\, a(x), b(y), c(z) \upharpoonright a(x) \prec b(y),\ b(y) \prec \mathbf{1},\ a(x) \prec c(z),\ c(z) \prec \mathbf{1} \,]$, more precisely.

[18] The restriction can be lifted with a trick. See §3.6.4.

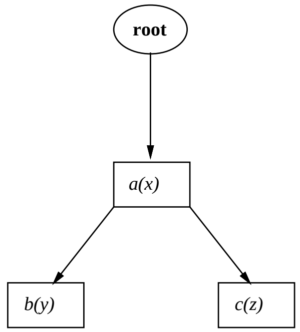


Figure 3.11. The guarding relation of $a(x) \prec b(y)$ and $a(x) \prec c(z)$

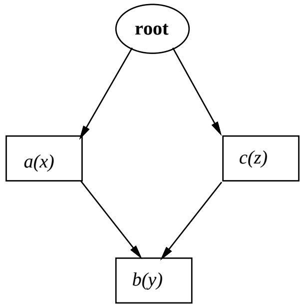


Figure 3.12. The guarding relation of $a(x) \prec b(y)$ and $c(z) \prec b(y)$

The obligatory guarding of mobile particles

In addition to the constraint, another constraint should be imposed on guarding in connection with mobility . As is observed in §2.4.2, the execution of mobile particles must be postponed until they are immobilised. Therefore, the particle mobilising a particle must guard it. When we declare the following molecules, for instance:

- $[\, \overline{a}\langle b\rangle, \overline{b}\langle c\rangle \,]$
- $[\, a(x), x(y) \,]$

we have to add to them the guarding such as:

- $[\, \overline{a}\langle b\rangle, \overline{b}\langle c\rangle \restriction \overline{a}\langle b\rangle \prec \overline{b}\langle c\rangle \,]$
- $[\, a(x), x(y) \restriction a(x) \prec x(y) \,]$

As we have seen in the previous chapter, the mobilising relations can be checked by depicting dataflow graphs. As is presented in the figure 3.13, the mobile particles are placed in lower levels than the particles mobilising them.

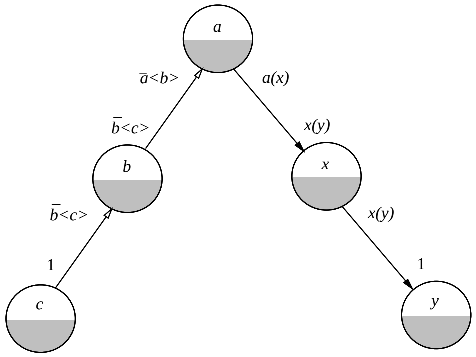


Figure 3.13. The dataflow graph composed of particles, $\overline{a}\langle b\rangle$, $\overline{b}\langle c\rangle$, $a(x)$, and $x(y)$

Notice, however, that the mobile particles need not be guarded directly by its mobilising particle, but other intermediate particles can be inserted in the guarding tree between them as long as it is reachable in the structure. For example, the expression:

$$[\, \overline{a}\langle b\rangle, \overline{d}\langle e\rangle, \overline{b}\langle c\rangle \restriction \overline{a}\langle b\rangle \prec \overline{d}\langle e\rangle,\ \overline{d}\langle e\rangle \prec \overline{b}\langle c\rangle \,]$$

respects the constraint as the relation, $\overline{a}\langle b\rangle \prec \overline{b}\langle c\rangle$, can be deduced owing to the transitive nature of guarding.

3.3.3 Restriction and the scope of bound names

To restrict access to names, we may declare the restriction in front of its molecule. That is, a process

$$(\nu\, b)(\overline{a}\langle b\rangle.\overline{c}\langle d\rangle.\mathbf{0})$$

is translated to a molecule

$$(\nu\, b)[\, \overline{a}\langle b\rangle, \overline{c}\langle d\rangle \restriction \overline{a}\langle b\rangle \prec \overline{c}\langle d\rangle \,]$$

The declaration also binds the name and determines its scope within the molecule. Analogously, the scope of bound names by input action is within the molecule, too. For example, when we translate a process such as:

$$a(x).c(y).\mathbf{0}$$

into:

$$[\, a(x), c(y) \restriction a(x) \prec c(y) \,]$$

the scopes of x and y, which are bound by input actions at a and c, respectively, are within the molecule.

Since we allow for nested molecules, the restriction does not need to be declared as to individual particles. For example, a process such as:

$$(\nu\, b)(\overline{a}\langle b\rangle) \ \mid a(x)$$

can be translated to:

$$[\, (\nu\, b)[\, \overline{a}\langle b\rangle \,], a(x) \,]$$

by turning $\overline{a}\langle b\rangle$ into a molecule.

3.3.4 Match

The match operator is treated in a similar way to guarding as its purpose is to guard particles. Observe in the following process that the match formula, $[x = b]$, can be regarded as guarding the particle, $c(y)$, by not allowing it to be executed until the condition is satisfied by substitution environments:

$$a(x).[x = b]c(y).\mathbf{0}$$

We will, therefore, add to the above expression the extra condition, $(x = b) \prec c(y)$:

$$[\, a(x), c(y) \restriction (x = b) \prec c(y),\ a(x) \prec c(y) \,]$$

The guarding by the match formula is not counted as the guarding by particles, thus any number of match formulas can guard a particle without modifying the tree structure of guarding relations.

3.3.5 The rule for actions

The dynamic semantics for $\mathcal{L}_0$ is defined with the rules specifying actions and interactions within and between molecules. The transition rule for actions specifies the conditions under which a particle can be executed. In short, a particle can be executed if it is not guarded by any particle or match formula. For example, a molecule such as $[\,\overline{a}\langle b\rangle, \overline{c}\langle d\rangle, \overline{e}\langle f\rangle\,]$ can degenerate to $[\,\overline{c}\langle d\rangle, \overline{e}\langle f\rangle\,]$ after executing $\overline{a}\langle b\rangle$, which is expressed as:

$$[\,\overline{a}\langle b\rangle, \overline{c}\langle d\rangle, \overline{e}\langle f\rangle\,] \;{}_{\overline{a}\langle b\rangle}\!\searrow\; [\,\overline{c}\langle d\rangle, \overline{e}\langle f\rangle\,]$$

Then, the molecule can degenerate further either to $[\,\overline{c}\langle d\rangle\,]$ or $[\,\overline{e}\langle f\rangle\,]$. Assume we would like to ensure $\overline{c}\langle d\rangle$ should be executed prior to $\overline{e}\langle f\rangle$. The execution can be controlled by adding to the molecule a constraint such that $\overline{c}\langle d\rangle \prec \overline{e}\langle f\rangle$:

$$[\,\overline{c}\langle d\rangle, \overline{e}\langle f\rangle \restriction \overline{c}\langle d\rangle \prec \overline{e}\langle f\rangle\,] \;{}_{\overline{c}\langle d\rangle}\!\searrow\; [\,\overline{e}\langle f\rangle\,]$$

The constraint, $\overline{c}\langle d\rangle \prec \overline{e}\langle f\rangle$, is eliminated once $\overline{c}\langle d\rangle$ is consumed because it is now void. Then, the remaining particle, $\overline{e}\langle f\rangle$, can be executed:

$$[\,\overline{e}\langle f\rangle\,] \;{}_{\overline{e}\langle f\rangle}\!\searrow\; \emptyset$$

To guard a particle with a match formula, we have to keep a record of substitutions as substitution environments, E. We may, for example, express a substitution environment $\{{}^{a}\!/_{x}\}$ as:

$$[\,a(y)\,] * \{{}^{a}\!/_{x}\}$$

where the expression, $*\{{}^{a}\!/_{x}\}$, following the molecule records the substitution, $a = x$. Given the record, we can control the execution with match by referring to the environment. For example, in a molecule with an environment below, the particle, $a(y)$, is executable:

$$[\,a(y) \restriction (a = x) \prec a(y)\,] * \{{}^{a}\!/_{x}\}\,{}_{a(y)}\!\searrow\; \emptyset$$

while it is not in a different environment below because the formula, $(b = x)$, is not satisfied by the environment, $\{{}^{a}\!/_{x}\}$.

$$[\,a(y) \restriction (b = x) \prec a(y)\,] * \{{}^{a}\!/_{x}\}$$

3.3.6 The rule for interactions

The interaction between an input and output particles may occur when both of them are executed at the same time, provided that they are connected with each other through a channel. Upon the interaction, the substitution environment is updated with the new substitution. For example, in the molecule below, $\overline{a}\langle b\rangle$ and $a(x)$ can interact with each other, updating the substitution environment E by $\{{}^{b}\!/_{x}\}$:

$$\begin{aligned}&[\,\overline{a}\langle b\rangle, a(x), \overline{c}\langle d\rangle\,] * E\\ &{}_{\tau}\!\searrow [\,\overline{c}\langle d\rangle\,] * E \cup \{{}^{b}\!/_{x}\}\end{aligned}$$

The substitution can apply to the molecule if the substituted name appears there. If $\overline{e}\langle x\rangle$ was in the molecule, it would have been rewritten to $\overline{e}\langle b\rangle$ as the result of the interaction as follows:

$$\begin{aligned}&[\,\overline{a}\langle b\rangle, a(x), \overline{c}\langle d\rangle, \overline{e}\langle x\rangle\,] * E\\ &{}_{\tau}\!\searrow [\,\overline{c}\langle d\rangle, \overline{e}\langle b\rangle\,] * E \cup \{{}^{b}\!/_{x}\}\end{aligned}$$

The substitution should apply to constraints, too. Suppose we impose a constraint, $a(x) \prec \overline{e}\langle x\rangle$, on particles in the molecule. Then, the constraint is rewritten as well: [19]

$$\begin{aligned}&[\,\overline{a}\langle b\rangle, a(x), \overline{c}\langle d\rangle, \overline{e}\langle x\rangle \restriction a(x) \prec \overline{e}\langle x\rangle\,] * E\\ &{}_{\tau}\!\searrow [\,\overline{c}\langle d\rangle, \overline{e}\langle b\rangle \restriction a(b) \prec \overline{e}\langle b\rangle\,] * E \cup \{{}^{b}\!/_{x}\}\end{aligned}$$

The same does not apply to match formulas, however. Suppose the output action, $\overline{e}\langle x\rangle$, is guarded by a match formula, $(b = x)$. Despite the fact that the name x is substituted by b, the formula remains to be the same as follows:

$$\begin{aligned}&[\,\overline{a}\langle b\rangle, a(x), \overline{c}\langle d\rangle, \overline{e}\langle x\rangle \restriction (b = x) \prec \overline{e}\langle x\rangle\,] * E\\ &{}_{\tau}\!\searrow [\,\overline{c}\langle d\rangle, \overline{e}\langle b\rangle \restriction (b = x) \prec \overline{e}\langle b\rangle\,] * E \cup \{{}^{b}\!/_{x}\}\end{aligned}$$

Scope intrusion and extrusion

As we have seen in the previous chapter, the interaction between particles and the effect of the restriction operator, ν , bring about *scope intrusion* and *extrusion*. Observe, for example, that the scope intrusion requires us to rename a in the first molecule below to a' to avoid that it crushes with a in the second molecule.

$$\begin{aligned}&[\,(\nu\, a)[\,b(x), \overline{a}\langle z\rangle\,], [\,\overline{b}\langle a\rangle, \overline{a}\langle c\rangle\,]\,] * E\\ &= [\,(\nu\, a')[\,b(x), \overline{a'}\langle z\rangle\,], [\,\overline{b}\langle a\rangle, \overline{a}\langle c\rangle\,]\,] * E\\ &{}_{\tau}\!\searrow [\,(\nu\, a')[\,\overline{a'}\langle z\rangle\,], [\,\overline{a}\langle c\rangle\,]\,] * E \cup \{{}^{a}\!/_{x}\}\end{aligned}$$

[19] Strictly speaking, the constraint would not appear as it is void.

As for scope extrusion, observe the next example, where the scope of a is extruded from the second molecule to the first upon the interaction:

$$[\,[\,b(x), x(z)\,], (\nu\, a)[\,\overline{b}\langle a\rangle, \overline{a}\langle y\rangle\,]\,] * E$$
$$\searrow_{\tau} [\,(\nu\, a)[\,a(z), \overline{a}\langle y\rangle\,]\,] * E \cup \{{}^{a}/_{x}\}$$

To define particular rules for scope intrusion and extrusion may complicate the semantics. Suppose we keep records on substitution environments as to each molecule in order to define a rule for scope intrusion, for example. The transition may occur as follows:

$$[\,(\nu\, a)[\,b(x), \overline{a}\langle z\rangle\,] * E_1, [\,\overline{b}\langle a\rangle, \overline{a}\langle c\rangle\,] * E_2\,]$$
$$= [\,(\nu\, a')[\,b(x), \overline{a'}\langle z\rangle\,] * E_1 \cup \{{}^{a'}/_{a}\}, [\,\overline{b}\langle a\rangle, \overline{a}\langle c\rangle\,] * E_2\,]$$
$$\searrow_{\tau} [\,(\nu\, a')[\,\overline{a'}\langle z\rangle\,] * E_1 \cup \{{}^{a'}/_{a}, {}^{a}/_{x}\}, [\,\overline{a}\langle c\rangle\,] * E_2\,]$$

This way of defining transitions is not particularly appealing. The rules may be complicated because the decision on which pair of particles will interact has to be made each time in advance to renaming particles. Nested molecules may make things more complicated, which are allowed for in our language.

An early renaming strategy

We treat the matter of bound names and their scopes based on a different strategy from that adopted in the π-calculus. We consider bound names as a convenient way to make them unique in their scopes. The scope extrusion is thought of as a way to preserve uniqueness of bound names, which will disappear once our strategy is adopted. Reconsider the timing when bound names are renamed in the π-calculus. Since they are not renamed until there is a crush, we call it *late renaming* strategy. Contrary to it, our strategy may be called *early renaming* strategy as we rename any bound names to unique ones in the beginning.

Suppose we would like to define a molecule such that two distinctive names are emitted from a channel, a. One way to do this is to use two distinctive names, i.e.,

$$[\,\overline{a}\langle b\rangle, \overline{a}\langle c\rangle\,]$$

But with binding force, we are released of the labour to assign two distinctive names to them. The following molecule does the same job as two occurences of b in the first and second molecules are distinctive owing to the effect of the restriction, ν.

$$[\,(\nu\, b)[\,\overline{a}\langle b\rangle\,], (\nu\, b)[\,\overline{a}\langle b\rangle\,]\,]$$

Similarly, a molecule such that it inputs two names and substitutes two distinctive names with them can be expressed as $[\,[\,a(x)\,], [\,a(x)\,]\,]$, not as $[\,a(x), a(y)\,]$.

By regarding binding names as a convenient way to introduce distinctive names, we *compile* the expressions containing bound names to those assigned unique names. Since we do not need to worry about the scope, we can dissolve all molecules into a single one. The easiest way to make a name unique is to mark it with consecutive natural numbers as subscripts. We may, therefore, compile the above program, $[\,(\nu\, b)[\,\overline{a}\langle b\rangle\,], (\nu\, b)[\,\overline{a}\langle b\rangle\,]\,]$, to $[\,\overline{a}\langle b_1\rangle, \overline{a}\langle b_2\rangle\,]$.[20] Given the compilation step, we do not need any rules for scope intrusion and extrusion. The step is also necessary to translate processes defined in the π-calculus to logical formulas because the logic cannot deal with the phenomena of scope extrusion and intrusion.

The syntax and semantics of $\mathcal{L}_0$ are summarised in Appendix A, where extended languages with the replication and choice operators are also defined.

3.4 The translation from $\mathcal{L}_0$ to C-ILL$_e^=$

3.4.1 The translation via dataflow graphs

The translation from $\mathcal{L}_0$ to C-**ILL**$_e^=$ is straightforward. Guardings are translated to the axioms of combinatorial linear logic just by replacing $\Rightarrow$ for $\prec$ and adding to it a term. For example, $\overline{c}\langle d\rangle \prec \overline{e}\langle f\rangle$ is translated to $\phi : \overline{c}\langle d\rangle \Rightarrow \overline{e}\langle f\rangle$. The match formulas are translated to formulas with '=', e.g., $(b = x) \prec \overline{e}\langle f\rangle$ is translated to $\psi : !\,(b = x) \Rightarrow \overline{e}\langle f\rangle$. By combining these translations, we can translate a process, $\overline{c}\langle d\rangle.[b = x]\overline{e}\langle f\rangle$, to $\varphi : \overline{c}\langle d\rangle \otimes\, !(b = x) \Rightarrow \overline{e}\langle f\rangle$.

The simple translation can, however, only capture actions, not interactions. To express possible interactions between processes in our logic, we have to represent particles in $\mathcal{L}_0$ as dataflow graph so that we can predict all possible interactions. We have already seen how a process in the π-calculus can be depicted as a dataflow graph (§3.1.5). Once a process is depicted as a dataflow graph, pairs of actions that can interact with each other can be easily predicted. By taking the top node as root in the figure, they are in the same depth or distance from the root. Observe, for example, how the depth is counted in the figure 3.14, which was shown originally as Figure 3.10 in the section.[21]

As we have seen in the section, the interactions, t_1 and t_2, are translated to the following:

(a) $t_1 : a(y) \otimes \overline{a}\langle b\rangle \Rightarrow\ y(z) \otimes \overline{b}\langle x\rangle \otimes\, !(b = y)$

[20] Recall we needed to rename them to depict dataflow graphs correctly (§2.1.2).

[21] We suppress subscripts where no confusion arises.

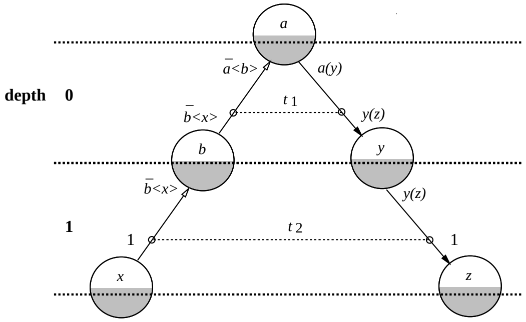


Figure 3.14. The dataflow graph decorated with actions

(b) $t_2 : y(z) \otimes \overline{b}\langle x\rangle \otimes\ !(b = y) \Rightarrow\ \mathbf{1} \otimes\ !(x = z)$

The general principle is that a pair of exporting and importing particles may interact with each other if

- they are in the same distance from a common node through which they are reachable, and
- they are located at the same place, i.e., they are connected through a channel.

The second condition is irrelevant when they are in the distance of 0, but it must be stated explicitly otherwise. Thus, we have to add the formula, $!(b = y)$, to the antecedent of t_2 to ensure that the pair of $\overline{b}\langle x\rangle$ and $y(z)$ should be connected through a channel, b.

One of our major inventions is certainly the graphical representation of particles. While it may be trivial to translate guardings into axioms of combinatory linear logic, it requires another step, dataflow graph representation, to detect possible interactions.

3.4.2 Extension from C-$\mathbf{ILL}_e$ to C-$\mathbf{ILL}_e^=$

Our logic, C-$\mathbf{ILL}_e^=$, is an extension of C-$\mathbf{ILL}_e$. As for axioms, we have added three axioms for substitutions. As for expressions, one can think that we have introduced to the logic three special predicates: `equal`, `export`, and `import`.

(a) The equal predicate, $=$, has two arguments a and b, where both must be a name. An expression such as $a = b$ is the abbreviation of a formula, $equal(a, b)$.

(b) The export predicate, $\overline{a}\langle b\rangle$, has two arguments, where both a and b must be a name. The first argument is a channel through which the second argument is exported. The expression, $\overline{a}\langle b\rangle$, is the abbreviation of a formula, $export(a, b)$.

(c) The import predicate, $a(b)$, has two arguments, where both a and b must be a name. The first argument is a channel through which a name is received to replace for the second argument. The expression, $a(b)$, is the abbreviation of a formula, $import(a, b)$.

One may recognise the logic is neither propositional nor first order. The additional predicates are, however, restricted to just these three.

3.4.3 Extension to the polyadic π-calculus

There is no problem in extending the language to treat the polyadic version of the π-calculus though we confined ourselves to the monadic version for simplicity. To express polyadic input or output particles in $\mathcal{L}_0$, we use the same notation as the polyadic π-calculus, e.g., $\overline{a}\langle b, c, d\rangle$, when we abbreviate it. To translate it fully into the logic, we simply increase the number of arguments. That is, $\overline{a}\langle b, c, d\rangle$ is to be translated to $export(a, b, c, d)$.

Accordingly, it will increase the number of formulas stating the effect on substitution environments upon interactions between polyadic particles. For example, if $\overline{a}\langle b, c, d\rangle$ and $a(x, y, z)$ interact with each other, then three formulas are needed to express the effect, e.g., $!(b = x), !(c = y)$, and $!(d = z)$. There is no difficulties in adding the formulas to axioms.

3.5 Non-determinism

We encounter two sort of non-determinism in the π-calculus, one of which is caused by the use of the choice operator, $+$, and the other of which occurs when there are more than one possible interactions, e.g., when more than one importing particle compete to get access to an exporting particle or when more than one exporting particles are available for an importing particle. We discuss the first case of non-determinism, then the second.

3.5.1 The choice operator as external non-determinism

The non-determinism by choice operator occurs for example when a process below defined in the π-calculus tries to output a name, b or c from channels y or z, respectively, after emitting x from a channel a.

$$\overline{a}\langle x\rangle.(\overline{y}\langle b\rangle + \overline{z}\langle c\rangle)$$

One way to express the non-determinism of the process is to translate the choice operator to additive conjunction, & (with). In that case, the above program may be translated into the below, where the non-deterministic choice is regarded as external one.

$$\overline{a}\langle x\rangle \Rightarrow \overline{y}\langle b\rangle \& \overline{z}\langle c\rangle$$

The translation is reasonable if we can understand the program as stating two possible transitions, one of which is to execute $\overline{a}\langle x\rangle$ then $\overline{y}\langle b\rangle$, and the other of which is to execute $\overline{a}\langle x\rangle$ then $\overline{z}\langle b\rangle$, where the decision is up to the process. The two possible transitions are expressed as axioms such as $\overline{a}\langle x\rangle \Rightarrow \overline{y}\langle b\rangle$, and $\overline{a}\langle x\rangle \Rightarrow \overline{z}\langle c\rangle$, respectively. By the inference rule of & (with), we can infer the above axiom as follows:

$$\frac{s : \overline{a}\langle x\rangle \Rightarrow \overline{y}\langle b\rangle \quad t : \overline{a}\langle x\rangle \Rightarrow \overline{z}\langle c\rangle}{\langle s,t\rangle : \overline{a}\langle x\rangle \Rightarrow \overline{y}\langle b\rangle \& \overline{z}\langle c\rangle}\ (\&)$$

Then the decision to choose either one is derived with the projection, π, as follows:[22]

$$\frac{\dfrac{s : \overline{a}\langle x\rangle \Rightarrow \overline{y}\langle b\rangle \quad t : \overline{a}\langle x\rangle \Rightarrow \overline{z}\langle c\rangle}{\langle s,t\rangle : \overline{a}\langle x\rangle \Rightarrow \overline{y}\langle b\rangle \& \overline{z}\langle c\rangle}\ (\&) \qquad \pi_0 : \overline{y}\langle b\rangle \& \overline{z}\langle c\rangle \Rightarrow \overline{y}\langle b\rangle}{\langle s,t\rangle \circ \pi_0 : \overline{a}\langle x\rangle \Rightarrow \overline{y}\langle b\rangle}\ (\circ)$$

Note that the expression, $\overline{a}\langle x\rangle \Rightarrow \overline{y}\langle b\rangle \& \overline{z}\langle c\rangle$ should not be confused with an expression with $\otimes$, $\overline{a}\langle x\rangle \Rightarrow \overline{y}\langle b\rangle \otimes \overline{z}\langle c\rangle$. While the former means the choice between the two particles, the latter means both can be executed. The latter, therefore, corresponds to a process such as:

$$\overline{a}\langle x\rangle.(\overline{y}\langle b\rangle \mid \overline{z}\langle c\rangle)$$

One may wonder how a case where a process starts by choosing one among others can be expressed. For example, the following process must start by choosing either one:

$$\overline{y}\langle b\rangle.\mathbf{0} + \overline{z}\langle c\rangle.\mathbf{0}$$

As we cannot find out any antecedent for them, the translation seems to fail. Recall, however, $\mathbf{1}$(unit) can appear anywhere. The process can therefore be rewritten to

$$\mathbf{1}.(\overline{y}\langle b\rangle + \overline{z}\langle c\rangle)$$

which enables us to translate it to the following formula using &:

$$\mathbf{1} \Rightarrow \overline{y}\langle b\rangle \& \overline{z}\langle c\rangle$$

Although such translation might appear to be cumbersome, there is no problem theoretically in the translation.

[22] The term, $\langle s,t\rangle \circ \pi_0$, is written as $\pi_0 \circ \langle s,t\rangle$ in standard notation, and reducible to s.

3.5.2 The choice operator as internal non-determinism

Another way to deal with choice is to regard it as expressing internal non-determinism, $\oplus$ (plus). If we adopt this approach, we have to modify the syntax of the π-calculus rather drastically so that the above program can be written as

$$(\overline{y}\langle b\rangle + \overline{z}\langle c\rangle).\mathbf{0}$$

Then, the program is translated to $\overline{y}\langle b\rangle \oplus \overline{z}\langle c\rangle \Rightarrow \mathbf{0}$, which is derived as follows:

$$\frac{u : \overline{y}\langle b\rangle \Rightarrow \mathbf{0} \quad v : \overline{z}\langle c\rangle \Rightarrow \mathbf{0}}{[u,v] : \overline{y}\langle b\rangle \oplus \overline{z}\langle c\rangle \Rightarrow \mathbf{0}}$$

In this case, it is appropriate to interpret the null process, $\mathbf{0}$, as $\mathbf{0}$ in linear logic, not as $\mathbf{1}$(unit) because we are supposed to interpret the meaning of choice as internal non-determinism, whose neutral element is $\mathbf{0}$. The trouble of this interpretation is that not all processes terminate; Some processes never terminate. Observe, for example, the behaviour of a process, $P + Q$, where $P =_{\mathrm{def}}\ !\overline{y}\langle b\rangle$ and $Q =_{\mathrm{def}}\ !\overline{z}\langle c\rangle$. As they repeat forever the actions, $\overline{y}\langle b\rangle$ and $\overline{z}\langle c\rangle$, respectively, they will never turn into the identical state, $\mathbf{0}$. Thus, we cannot express choice as internal non-determinism.

To sum up, it is impossible to interpret choice as internal non-determinism without altering the π-calculus. We have to ensure both options should be finite, which seems to be too restrictive as a translation of the π-calculus. We prefer, therefore, interpreting choice as external non-determinism.

3.5.3 Non-deterministic interaction

Another sort of non-determinism can occur in evolution, for example, when two importing particles compete to get access to an exporting particle. Observe the following system where the name b may substitute either x in the second process or y in the third process non-deterministically:

$$\overline{a}\langle b\rangle.P \mid a(x).Q \mid a(y).R$$

If the second process gets access to the first process, then the system turns into the following state:

$$P \mid Q\{{}^{b}/_{x}\} \mid a(y).R$$

While the system turns into another one shown below if the third one gets access:

$$P \mid a(x).Q \mid R\{{}^{b}/_{y}\}$$

This kind of non-deterministic interaction can be captured as follows: We first turn the process into a graph shown in the figure 3.15, then translate it into the following linear logical formulas:

(a) $p : \overline{a}\langle b\rangle \Rightarrow P$

(b) $q : a(x) \Rightarrow Q$

(c) $r : a(y) \Rightarrow R$

(d) $s : \overline{a}\langle b\rangle \otimes a(x) \Rightarrow !(b = x) \otimes P \otimes Q$

(e) $t : \overline{a}\langle b\rangle \otimes a(y) \Rightarrow !(b = y) \otimes P \otimes R$

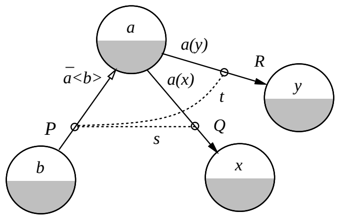


Figure 3.15. A non-deterministic interaction

Then, the non-deterministic interaction can be derived as below:[23]

$$\dfrac{\dfrac{s : \overline{a}\langle b\rangle \otimes a(x) \Rightarrow !(b = x) \otimes P \otimes Q \quad id : a(y) \Rightarrow a(y)}{s \otimes id : \overline{a}\langle b\rangle \otimes a(x) \otimes a(y) \Rightarrow !(b = x) \otimes P \otimes Q \otimes a(y)} \quad \dfrac{t : \overline{a}\langle b\rangle \otimes a(y) \Rightarrow !(b = y) \otimes P \otimes R \quad id : a(x) \Rightarrow a(x)}{t \otimes id : \overline{a}\langle b\rangle \otimes a(x) \otimes a(y) \Rightarrow !(b = y) \otimes P \otimes R \otimes a(x)}}{\langle s \otimes id, t \otimes id\rangle : \overline{a}\langle b\rangle \otimes a(x) \otimes a(y) \Rightarrow (!(b = x) \otimes P \otimes Q \otimes a(y))\ \&\ (!(b = y) \otimes P \otimes R \otimes a(x))}$$

The decision to choose either one is derived with the projection, π, as is explained in §3.5.1. It is pleasant to see that both kind of non-determinism, one by choice operator and the other by non-deterministic interaction, can fall into the same sort of non-determinism, external non-determinism.

3.6 Discussion

3.6.1 Replication

There are of course limitations to our specification. First of all, we cannot capture the behaviour of processes with replication, !. Consider how a process defined in the

[23] In the right part of the proof, we skip for simplicity the step to swap $a(y)$ and $a(x)$, i.e., turning $a(y) \otimes a(x)$ into $a(x) \otimes a(y)$.

π-calculus, $!\,\overline{a}\langle b\rangle$, may be expressed in our logic. If the action is performed only once, it would be expressed as:

$t_1 : \overline{a}\langle b\rangle \Rightarrow \mathbf{1}$

If the action is performed twice, then it should be expressed as:

$t_1 : \overline{a}\langle b\rangle \Rightarrow \mathbf{1}$

$t_2 : \overline{a}\langle b\rangle \Rightarrow \mathbf{1}$

This suggest we have to consider how many copies are needed when we compile processes to logical formulas. The solution works as long as the number of interactions is bound to be finite. For example, in the case below we know the particle, $\overline{a}\langle b\rangle$, should be copied twice so that both importing particles, $a(x)$ and $a(y)$, can receive the name, b.

$!\,\overline{a}\langle b\rangle \mid a(x) \mid a(y)$

But the solution cannot be applied to the case where both exporting and importing particles are replicable. There is no way to count how many copies are required in the following case:

$!\,\overline{a}\langle b\rangle \mid !\,a(x) \mid !\,a(y)$

A better solution is therefore to provide for an extra-logical mechanism generating a new axiom when it is needed. Recall that neither Hennessy-Milner logic [HM85] nor its extention to mobile processes [MPW91] does not deal with the issue and that fixed-point operators had to be introduced to express properties such as "performing some actions forever" [Sti92]. The fact suggests that we have to extend the logic rather drastically to incorporate replication into it.

Another approach is to admit axioms to be used as many times as needed. In this case we have to give up with the view that terms correspond to a particular action, but to a more abstract computation. Even the approach still can see a trouble when replication is combined with restriction. To specify a process such as $!(\nu\, b)(\overline{a}\langle b\rangle)$, finitely many b_i are needed as b must be a fresh name each time it is created, which cannot be coped with by the approach either. Only finite cases can be dealt with. This is also the case for petri nets; Only finite petri nets could be investigated by linear logic [MOM91].

3.6.2 Bound and free names

We have not investigated the issue of bound/free names. In our treatment, bound names are assigned unique names when processes are translated into logical formulas so that they will never crash with each other. This might expose another limitation when the restriction operator, ν , is combined with the replication, !, as we have seen in the above. To solve the problem, one may think of introducing to the logic existential quantifier, $\exists$. For example, one may try to express a process $(\nu\, b)(\overline{a}\langle b\rangle)$ as

$$s_1 : \exists b.\overline{a}\langle b\rangle \Rightarrow \mathbf{1}$$

The solution will however not work because the scope of the quantifier is limited to within the formula. Observe for example how the following process can be translated to a set of formulas:

$$!\,(\nu\, b)\overline{a}\langle b\rangle.\overline{b}\langle c\rangle.\overline{b}\langle d\rangle.\mathbf{0}$$

which can be translated into the following set of formulas, where the name, b, is bound:

$$u_1 : \exists b.\overline{a}\langle b\rangle \Rightarrow \overline{b}\langle c\rangle$$
$$v_1 : \exists b.\overline{b}\langle c\rangle \Rightarrow \overline{b}\langle d\rangle$$
$$w_1 : \exists b.\overline{b}\langle d\rangle \Rightarrow \mathbf{1}$$

The problem is that the scope of the existential quantifier does not expand beyond an axiom, and therefore the names, b, are not identical between these axioms. Thus, the approach falls short of capturing the meaning of bound names. In fact, we need to introduce more powerful operators to deal with the problem than those provided for in standard logics. Dam for example adopt the λ-abstraction and application [Dam93]. With more powerful operations, *abstraction* and *concretion*, discussed in [Mil93b], we may express those axioms as:

$$u_1 : \langle\rangle(\overline{a}\langle b\rangle \Rightarrow \overline{b}\langle c\rangle)[b]$$
$$v_1 : \langle b\rangle(\overline{b}\langle c\rangle \Rightarrow \overline{b}\langle d\rangle)[b]$$
$$w_1 : \langle b\rangle(\overline{b}\langle d\rangle \Rightarrow \mathbf{1})[\,]$$

where $\langle b\rangle$ preceeding to the formula abstracts it over b, and $[b]$ following to the axiom exports the name to another formula combined with it. Given these axioms, we can successfully import b from u_1 to v_1 as follows:

$$\frac{u_1 : \langle\rangle(\overline{a}\langle b\rangle \Rightarrow \overline{b}\langle c\rangle)[b] \quad v_1 : \langle b\rangle(\overline{b}\langle c\rangle \Rightarrow \overline{b}\langle d\rangle)[b]}{u_1 \circ v_1 : \langle\rangle(\overline{a}\langle b\rangle \Rightarrow \overline{b}\langle d\rangle)[b]}$$

Pursuing the idea further leads to an algebraic theory, *action structures* [Mil92b, Mil93a, Mil95]. Such an investigation goes beyond the scope of the thesis, however.

3.6.3 Temporal properties

We have seen that interactions can be controlled by imposing a constraint on substitution environments (§2.4.3). For example, there are six possible cases of interactions in the following system:

$$\overline{a}\langle b\rangle \;|\; \overline{a}\langle c\rangle \;|\; a(x) \;|\; a(y)$$

(a) between $\overline{a}\langle b\rangle$ and $a(x)$,

(b) between $\overline{a}\langle b\rangle$ and $a(y)$,

(c) between $\overline{a}\langle c\rangle$ and $a(x)$,

(d) between $\overline{a}\langle c\rangle$ and $a(y)$,

(e) between $\overline{a}\langle b\rangle$ and $a(x)$, and between $\overline{a}\langle c\rangle$ and $a(y)$, or

(f) between $\overline{a}\langle b\rangle$ and $a(y)$, and between $\overline{a}\langle c\rangle$ and $a(x)$

If we impose a constraint such that b should substitute x eventually, only the first and fifth interactions are allowed to occur. That sort of control goes beyond the ability of the logic and may be considered as a meta-level inference in the sense that proofs are regarded as objects. Let us see how the example can be treated in our logic. The actions and interactions observable in the system are specified with the following set of axioms:

(a) $t : \overline{a}\langle b\rangle \Rightarrow \mathbf{1}$

(b) $u : \overline{a}\langle c\rangle \Rightarrow \mathbf{1}$

(c) $v : a(x) \Rightarrow \mathbf{1}$

(d) $w : a(y) \Rightarrow \mathbf{1}$

(e) $p : \overline{a}\langle b\rangle \otimes a(x) \Rightarrow !(b = x) \otimes \mathbf{1}$

(f) $q : \overline{a}\langle b\rangle \otimes a(y) \Rightarrow !(b = y) \otimes \mathbf{1}$

(g) $r : \overline{a}\langle c\rangle \otimes a(x) \Rightarrow !(c = x) \otimes \mathbf{1}$

(h) $s : \overline{a}\langle c\rangle \otimes a(y) \Rightarrow !(c = y) \otimes \mathbf{1}$

Then, the admissible derivations are the axiom p itself and the following one composed of p and s.[24] Other axioms, q, r, and s, and the theorem inferred from q and r, are not admissible.

$$\frac{p : \overline{a}\langle b\rangle \otimes a(x) \Rightarrow !(b = x) \otimes \mathbf{1} \quad s : \overline{a}\langle c\rangle \otimes a(y) \Rightarrow !(c = y) \otimes \mathbf{1}}{p \otimes s : \overline{a}\langle b\rangle \otimes a(x) \otimes \overline{a}\langle c\rangle \otimes a(y) \Rightarrow !(b = x) \otimes !(c = y) \otimes \mathbf{1}}$$

[24] We skip the step to turn $\mathbf{1} \otimes \mathbf{1}$ into $\mathbf{1}$ and another one to swap $\mathbf{1}$ and $!(c = y)$, for simplicity.

The observation suggests that temporal properties are computationally expensive. In fact, *liveness* and *safety* are expressed by introducing a fixed-point operator in the modal mu-calculus [Sti92]. It is not certain whether or not introducing fixed-points is the only solution, but the issue should certainly be investigated in a higher order level than first order.

3.6.4 The restriction on guarding relations

We have seen that guarding relations must form a tree to be faithful to the π-calculus(§3.3.2). The first molecule of the followings for example is admissible, while the second is not since $b(y)$ is guarded by two particles, $a(x)$ and $c(z)$:

(a) $[\, a(x), b(y), c(z) \upharpoonright a(x) \prec b(y),\ a(x) \prec c(z) \,]$

(b) $[\, a(x), b(y), c(z) \upharpoonright a(x) \prec b(y),\ c(z) \prec b(y) \,]$

The restriction can, however, be lifted. Let us first examine where the problem is. What we would like to mean by the constraints, "$\upharpoonright a(x) \prec b(y),\ c(z) \prec b(y)$", is that the particle $b(y)$ should be ready only when *both* $a(x)$ and $c(z)$ have been executed. If we translate these guarding formulas straightforwardly into linear logic formulas, they would be translated to the followings, respectively, where we express the third one explicitly.

- $\phi : a(x) \Rightarrow b(y)$
- $\psi : c(z) \Rightarrow b(y)$
- $\varphi : b(y) \Rightarrow \mathbf{1}$

The problem of this translation is that it alters the process; it means that $b(y)$ can be executed *either* when $a(x)$ or $c(z)$ has been executed.[25] To avoid the problem, we can modify the antecedent of the third formula by duplicating the particle as follows:

- $\phi : a(x) \Rightarrow b(y)$
- $\psi : c(z) \Rightarrow b(y)$
- $\varphi : b(y) \otimes b(y) \Rightarrow \mathbf{1}$

Since the axiom, φ, requires two occurences of $b(y)$, it can only be applied to when ϕ and ψ are combined by parallel composition to compose a formula.

Similar asymmetry can also be found in translating the constraint, $\upharpoonright a(x) \prec b(y), a(x) \prec c(z)$, to logical formulas. To mean that both $b(y)$ and $c(z)$ are executable after $a(x)$ is executed, one has to express it as $a(x) \Rightarrow b(y) \otimes c(z)$. We have solved the problem by decorating dataflow graphs using the multiplicative connective, $\otimes$, not the additive, &.

[25] This is because $[\phi, \psi] : a(x) \oplus c(z) \Rightarrow b(y)$ is constructed out of ϕ and ψ as coproduct in the logic.

3.6.5 Comparison with Hennessey-Milner Logic

An improvement of the logic to Hennessey-Milner logic (HML) [HM85] is that we can express internal structures of interaction. Observe how transitions occuring between the processes shown below may be expressed in HML, which was presented in the figures of 3.10 and 3.14,

$$\begin{array}{rr|l} & \overline{a}\langle b\rangle.\overline{b}\langle x\rangle.\mathbf{0} & a(y).y(z).\mathbf{0} \\ \xrightarrow{\tau} & \overline{b}\langle x\rangle.\mathbf{0} & b(z).\mathbf{0}\{{}^{b}\!/_{y}\} \\ \xrightarrow{\tau} & \mathbf{0} & \mathbf{0}\{{}^{b}\!/_{y}, {}^{x}\!/_{z}\} \end{array}$$

As the transitions are invisible from outside, it is specified by the below formula, which means the process may end up with **true** after performing the two consecutive silent actions, τ. The substitution environment is expressed as an extra condition:

$$P \models \langle\tau\rangle\langle\tau\rangle\mathbf{true}, \text{ where } \{{}^{b}\!/_{y}, {}^{x}\!/_{z}\}$$

On the other hand, the logic can trace the steps by which processes turn into a particular state. The same transitions can be described in the logic as follows, which has been shown in §3.1.5:

$$\dfrac{t_1 : a(y) \otimes \overline{a}\langle b\rangle \Rightarrow y(z) \otimes \overline{b}\langle x\rangle \otimes\, !(b=y) \qquad \dfrac{\dfrac{t_2 : y(z) \otimes \overline{b}\langle x\rangle \otimes\, !(b=y) \Rightarrow \mathbf{1} \otimes\, !(x=z) \qquad \dfrac{id : (b=y) \Rightarrow (b=y)}{!id : !(b=y) \Rightarrow !(b=y)}\,!}{t_2 \otimes !id : y(z) \otimes \overline{b}\langle x\rangle \otimes\, !(b=y) \otimes\, !(b=y) \Rightarrow \mathbf{1} \otimes\, !(x=z) \otimes\, !(b=y)}\,\otimes}{t_2 \otimes !id : y(z) \otimes \overline{b}\langle x\rangle \otimes\, !(b=y) \Rightarrow \mathbf{1} \otimes\, !(x=z) \otimes\, !(b=y)}\,con}{t_1 \circ (t_2 \otimes !id) : a(y) \otimes \overline{a}\langle b\rangle \Rightarrow \mathbf{1} \otimes\, !(x=z) \otimes\, !(b=y)}\,cut$$

In the derivation, the information about how interactions occured is stored in the term, $t_1 \circ (t_2 \otimes !id)$, which means there was first an interaction, t_1, then t_2.[26] We can also see that the system was initially in the state such that it is ready to perform two particles, $a(y)$ and $\overline{a}\langle b\rangle$, concurrently, and that it ended up with the state such that there is no more possible actions with a substitution environment, ${}^{x}\!/_{z}$ and ${}^{b}\!/_{y}$. The silent action, τ, may be captured by defining term reduction as if it were $\mathbf{1}$.

What we cannot express in the logic is however negative information. HML can for example specify a process such that $\overline{a}\langle b\rangle$ is never followed by a silent action as [27]

$$P \models [\overline{a}\langle b\rangle][\tau]\mathbf{false}, \text{ where } \{{}^{b}\!/_{y}, {}^{x}\!/_{z}\}$$

[26] This is because the term can be reduced to $t_1 \circ t_2$

[27] This is because $a(y)$ must be performed before the interaction between $\overline{b}\langle x\rangle$ and $y(z)$ occurs. Another formula is however legitimate, the formula that $P \models \langle\overline{a}\langle b\rangle\rangle\langle a(y)\rangle\langle\tau\rangle\mathbf{true}$.

The **false** in HML is classical in the sense it is defined as unprovability of **true**. As our logic is an intuitionistic and constructive, we cannot deal with HML's **false** without modification to the logic. If we choose to adopt the classical negation, we have to extend the logic to classical one by introducing multiplicative disjunction, $⅋$, the dual of $\otimes$, with its neutral element, $\perp$. Then, $\neg P$ is defined as $P \multimap \perp$. On the other hand, if we remain within intuitionistic logic, we define it as $P \multimap \mathbf{0}$, which is adopted in Girard's translation of intuitionistic logic to linear logic [Gir87]. [28] We would like to adopt the latter approach and to remain within intuitionistic logic because Situation Theory has to do with intuitionistic logic as is observed in [BE90].

The other problem is that the logic is so powerful that it can derive unwanted derivations, too, when used to specify processes. See for example with the following set of axioms, we can infer not only $h \circ i : a(t) \Rightarrow \mathbf{1}$ but also $h \otimes i : a(y) \otimes y(z) \Rightarrow y(z) \otimes \mathbf{1}$ by parallel composition, which obviously we would like to avoid deriving.

(a) $h : a(y) \Rightarrow y(z)$

(b) $i : y(z) \Rightarrow \mathbf{1}$

This is because the motivation to apply linear logic to concurrency, i.e., originally to petri nets, was to investigate reachability problem, e.g., whether a state change is possible for a net. The solution taken by HML to constrain inference, if we try to find any, is to introduce modality to the logic. In HML, we can ensure the particle $a(y)$ should be executed before $y(z)$ by imposing an order on execution as a modality such as $\langle a(y)\rangle\langle y(z)\rangle$**true**. If we persue that direction, we may end up with something similar to the work by Dam [Dam94], a modal linear logic. We try to introduce a form of modality into our logic, which will be explained in §4.5.2. But there might be another way to construct formulas under control. Recall the formulas in HML specify labelled transition systems (LTS). Since LTSs can be thought to correspond to proof trees in the logic, we can regard the modal information encoded in HML to operate on proofs. Given this view, we could apply meta-programming technique to control theorem provers. But such investigation goes beyond the scope of the thesis.

3.6.6 The π-calculus specified in linear logic

It is not the first attempt to capture the computation by π-calculus using linear logic. There are at least two works by Miller [Mil92a] and Okada [Oka93], but their aims and approaches are different from ours. Miller starts by examining classical linear logic and maps each rule to that of the π-calculus. After these mappings have been done, he

[28] There is also another proposal to simulate classical negation within intuitionistic linear logic [EW93], where they define $\neg P$ as $P \& \mathbf{1} \multimap \mathbf{0}$ with other additional axioms.

adds to the logic particular rules to capture the effects brought about in the π-calculus because the only rules mapped from the calculus fall short of capturing them fully. Additional rules are introduced to encode guarding, choice, replication, and match, which go beyond linear logic. Okada pursuits that approach further and devises a number of axioms and rules so that asynchronous and synchronous communication can be captured as well as mobility. Higher order process calculi, where even processes can be exchanged, are also studied in the framework.

While they are interested in enriching linear logic by adding to it a number of axioms and rules so that the full-version of the π-calculus can be expressed within it, we are interested in finding out an appropriate computational ground for communication and concurrency for our purpose rather than simulating a particular language. That is why we have chosen combinatory intuitionistic linear logic because it can be related with a basic category, i.e., *symmetric monoidal* category with *product*. Through translation of processes in the π-calculus to formulas in the logic, we could understand the nature of non-determinism in the calculus.

The table 3.7 shows equations for the categories, which is called *intuitionistic linear category* by Troelstra [Tro92]. The category axioms and functional character of $\otimes$ with natural isomorphisms for γ, α, and λ define a symmetric monoidal category. The equations for $x \multimap -$ make the category closed. The last two columns shows equations for products and coproducts. We do not include equations for the exponentials and coherence conditions into the table for simplicity as we do not go into the discussion on categories.[29] We also adopt standard notation as for '$\circ$', i.e., $\phi \circ \psi$ reads in standard manner.

3.7 Conclusion

In this chapter, we have proposed a way to specify mobile processes using a combinatory intuitionistic linear logic with additional axioms for substitutions and three special predicates. We have shown how the processes defined in the π-calculus can be specified in the logic; they are first translated into a language $\mathcal{L}_0$, a syntactic variant of the calculus, then depicted as a dataflow graph to predict possible interactions, and finally expressed as formulas of the logic. While we could not specify the programs with replication, !, and restriction, ν, in the logic, its computational model is kept simple: a symmetric monoidal category (SMC) with product. Through discussion, we have clarified the limitation of the logic and what to be done towards specifying the full

[29] Troelstra's text or Bierman's paper [Bie95] is helpful to know the issue in more detail.
[30] γ means interchange of factors in a product. It is, therefore, self-inverse, i.e., $\gamma^{-1} = \gamma$.

arrows	ϕ, ψ, χ
identity	$id_A : A \longrightarrow A$
natural transformations	$\gamma_{A,B} : A \otimes B \longrightarrow B \otimes A$ $\alpha_{A,B,C} : A \otimes (B \otimes C) \longrightarrow (A \otimes B) \otimes C$ $\lambda_A : \mathbf{1} \otimes A \longrightarrow A$ if $\phi : A \otimes B \longrightarrow C$ then $cur_{A,B,C(\phi)} : A \longrightarrow (B \multimap C)$
category axioms	$(\phi \circ \psi) \circ \chi = \phi \circ (\psi \circ \chi)$ $id \circ \phi = \phi$ $\phi \circ id = \phi$
functional character of $\otimes$	$(\phi \circ \phi') \otimes (\psi \circ \psi') = (\phi \otimes \psi) \circ (\phi' \otimes \psi')$ $id \otimes id = id$ $1 = id_{\mathbf{1}}$
natural isomorphisms for γ, α, and λ	$\gamma \circ (\phi \otimes \psi) = (\psi \otimes \phi) \circ \gamma$ $\gamma \circ \gamma = id$ [30] $((\phi \otimes \psi) \otimes \chi) \circ \alpha = \alpha \circ (\phi \otimes (\psi \otimes \chi))$ $\alpha \circ \alpha^{-1} = id$ $\alpha^{-1} \circ \alpha = id$ $\lambda \circ (1 \otimes \phi) = \phi \circ \lambda$ $\lambda \circ \lambda^{-1} = id$ $\lambda^{-1} \circ \lambda = id$
Equations for $x \multimap -$	$cur(\phi) \circ \psi = cur(\phi \circ (\psi \otimes id))$ $ev \circ (cur(\phi) \otimes \psi) = \phi \circ (id \otimes \psi)$ $cur(ev) = id : (A \multimap B) \longrightarrow (A \multimap B)$
Equations for products	$\pi_1 \circ \langle \phi, \psi \rangle = \phi$ $\pi_2 \circ \langle \phi, \psi \rangle = \psi$ $\langle \phi, \psi \rangle \circ \chi = \langle \phi \circ \chi, \psi \circ \chi \rangle$ $\langle \pi_1, \pi_2 \rangle = id_{A \& B}$ $\top_A \circ \phi = \top_B : B \longrightarrow \top$ if $\phi : B \longrightarrow A$ $\top_\top = id_\top$
Equations for coproducts:	$[\phi, \psi] \circ \kappa_1 = \phi$ $[\phi, \psi] \circ \kappa_2 = \psi$ $\chi \circ [\phi, \psi] = [\chi \circ \phi, \chi \circ \psi]$ $[\kappa_1, \kappa_2] = id_{A \oplus B}$ $\phi \circ \bot_A = \bot_B : \bot \longrightarrow B$ if $\phi : A \longrightarrow B$ $\bot_\bot = id_\bot$

Table 3.7. Equations for intuitionistic linear categories

version of the π-calculus.

Chapter 4

Representations as processes

4.1 Introduction

In this chapter, we base Situation Semantics on a computational ground of communication and concurrency. One of the core ideas of Situation Semantics is *ecological realism*, the idea that meaning arises from the interaction between a cognitive agent and his or her environment. We model both the agent and environment as a process and study the interaction as a system of communicating processes. To elaborate the idea, we turn to the π-calculus [MPW92] and construct semantic objects employed in Situation Semantic [BC93] as processes. The construction helps us relate Situation Semantics with Linear Logic, through our translation of the calculus to Combinatory Intuitionistic Linear Logic.

4.1.1 Computational aspects of Situation Semantics

A lot of work has already been carried out on the foundation of Situation Semantics. For instance, Non-Well-Founded Set Theory [Acz88] and Aczel-Lunnon universe [AL91] provide for a basis to study semantic objects employed in Situation Semantics. While the aim of the investigation is to define a mathematical universe in which semantic objects can be properly constructed, we start by assuming the existence of a proper universe and study computational aspects of Situation Semantics. By "computational aspects", we mean the following issues:

- Given the expression of a semantic object and a model, how can one prove that it is the model for the expression?
- Given the expression of a semantic object, how can one prove that there exists a model for it?

- How can semantic objects and models be represented to investigate these issues?

To answer to the first and second questions, we have to answer to the third question. Our approach is to represent semantic objects employed in Situation Semantics as a theory of the logic presented in the previous chapter. Once we succeed in establishing the connection between Situation Semantics and the logic, the questions can be answered thanks to the result obtained through the investigation into linear logic.

4.1.2 Channel Theory and Situation Semantics

Another aim of the investigation is to relate Channel Theory [Bar92, Bar93, BS92, SB93] to Situation Semantics. The theory studies how information can be conveyed from one to another and refines the notion of *constraint* in Situation Theory. A number of notions have been invented and studied, but it is yet to be clarified how the result can be imported to and utilised in Situation Semantics.

The work relating Situation Semantics with Linear Logic through the π-calculus suggests one way to fill the gap between Channel Theory and Situation Semantics, especially Situation Theoretic Discourse Representation Theory [Coo93a, Coo93b], which have been hitherto studied rather independently. Intuitionistic linear category [Tro92] can actually cover Barwise's postulates on the principle of information flow [Bar93]. We will conclude the chapter by discussing the issue.

4.1.3 Representing situation-theoretic objects in linear logic

We sketch here our strategy to represent situation-theoretic objects in linear logic. We translate the connectives employed in Situation Semantics, conjunction, $\wedge$, disjunction, $\vee$, to &, and $\oplus$ in the logic, respectively, hinted by Girard's translation of intuitionistic logic into linear logic (Table 4.1) [Gir87]. Constraints are in the beginning translated to $\Rightarrow$, but later to $\multimap$ to deal with concepts belonging to higher order such as quantification.

Intuitionistic Logic		Linear Logic
p^*	:=	p (p atomic)
$(A \Rightarrow B)^*$	:=	$!A^* \multimap B^*$
$(A \wedge B)^*$	:=	$A^* \& B^*$
$(A \vee B)^*$	:=	$!A^* \oplus !B^*$
$(\neg A)^*$	:=	$!A^* \multimap \mathbf{0}$

* is translation function

Table 4.1. Girard's translation of intuitionistic logic to linear logic

We adopt the translation of negation as well to make our logic intuitionistic. We can alternatively use $\perp$, the neutral element of multiplicative disjunction, $\invamp$, for $\mathbf{0}$ to make the logic classical, but we prefer intuitionistic negation to classical one because we think it is more appropriate to model the theory of infons. After these mappings, we are still left with multiplicative conjunction, $\otimes$. We exploit the freedom to build various situation-theoretic objects such as infons and types.

4.1.4 Constructing situation-theoretic objects in the π-calculus

To understand underlying ideas in representing situation-theoretic objects in linear logic, it is helpful to encode them into the π-calculus although it lacks the ability to express disjunctive information, say $\oplus$. We therefore encode them into the π-calculus where applicable. We may even skip the translation to linear logic for simplicity because the correspondence has already been explained in the previous chapter. Not only the π-calculus, but also molecular language may we sometimes employ, e.g., when encoding simultaneous abstraction, because the use of guarding in the calculus is too restrictive to encode some objects.

In the chapter, we consider two versions of Situation Theory, primitive and naive. The first version is a simplified Situation Theory to explain our ideas in constructing situation-theoretic objects as processes. Some part of the construction of primitive situation-theoretic objects has been presented by the author in [Fuj94].

4.1.5 Related work

The project is by no means the first attempt to construct semantic objects in a type-theoretic framework. Some type theories have already been used to represent semantic objects employed in semantics for natural language. Ranta for example adopted Martin-Löf's Type Theory and analysed 'donkey' anaphora using dependent types [Ran91]. René Ahn [Ahn94] proposed to use Pure Type Systems as a "Language of Thought" to model dynamic knowledge states and argued that the system could be seen as a generalisation of Discourse Representation Structures (DRSs). Muskens has employed the first order part of ordinary type logic to express DRSs [Mus93, Mus94].

Their approaches and systems employed are fairly different, but they share the view that type theory may provide for fine-grained tools to study the semantics of natural language than the first order logic does. It is also expected that type theoretical foundation might enable us to study computational issues in more detail owing to the connection with computer science. There must be a number of ways in exploiting the possibility. Our project is another attempt in the area.

4.2 Primitive Situation Theory

We review semantic objects employed in primitive Situation Semantics. It is primitive in that we do not include *relations* and situation-theoretic *types*. We include them in naive Situation Semantics and postpone discussion related to those complex objects until the section §4.4. The reason why we do not include relations and types in the primitive situation semantics is that we would like to start with as simple theory as possible to make it easier to understand our ideas. Quantification is also discussed later in §4.6.

We start by explaining about *infon*, *situation*, and *proposition*. These notions form the core of Situation Semantics. To explain them informally, infons are items of information, and situations are objects that may or may not support them. Infons and situations consist of *propositions* such that a situation supports an infon. We then review the objects obtained through *abstractions*. By incorporating parameters into objects and abstracting them over parameters, we obtain *infon abstracts* and *proposition abstracts*. The operation to assign concrete objects to parameters is called *anchoring*. These abstract objects are turned into concrete objects through the operation.

4.2.1 Basic infons

We adopt mainly the Extended Kamp Notation (EKN) [BC91, BC93] to explain about semantic objects employed in Situation Semantics, along with linear notation. In EKN, the information such as *"John smiles"* is expressed as is shown in (4.1), where 'j' of smile(j) denotes the person who is called *"John"* and smiles. The predicate, 'smile', is called a *relation* and 'j' an *argument*. The relation whose number of argument is just one may sometimes be called *property*. It may also be expressed as (4.2) in linear notation previously used in the literature. The number '1' appearing in the expression indicates that the information is positive.

$$\boxed{\text{smile(j)}} \tag{4.1}$$

$$\langle\!\langle \text{smile, j; 1} \rangle\!\rangle \tag{4.2}$$

Similarly, the negative information such as *"John does not smile"* is expressed as is shown in (4.3) or (4.4). The symbol, '$\neg$', in (4.3) and the digit number, 0, in (4.4) indicate the information is negative. The indication of positive or negative is called *polarity.* The reason for explicitly referring to the polarity comes from our motivation to capture the meaning of natural language sentences. If one knows that John does not smile, he knows the fact. Such an item of knowledge should be distinguished from his lack of knowledge about John's physical movement. If we regarded negative information simply as lack of knowledge, then we would fail to capture the intuitive meaning of the sentence.

$$\boxed{\neg\text{smile(j)}} \tag{4.3}$$

$$\langle\langle \text{ smile, j; 0}\rangle\rangle \tag{4.4}$$

4.2.2 Conjunctive and disjunctive infons

The basic infons can be combined with connectives, $\wedge$ and $\vee$, to represent more complex items of information. Suppose we would like to express an item of information such as *"John smiles and Mary sings."* Then, it may be expressed either as (4.5) in EKN notation or as (4.6) in linear notation.

$$\boxed{\begin{array}{l}\text{smile(j)}\\ \text{sing(m)}\end{array}} \tag{4.5}$$

$$\langle\langle \text{ smile, j; 1}\rangle\rangle \wedge \langle\langle \text{ sing, m; 1}\rangle\rangle \tag{4.6}$$

Basic infons may also be combined disjunctively. We may express an item of information such as *"John smiles* **or** *Mary sings."* either as (4.7) or (4.8).

$$\boxed{\text{smile(j)} \vee \text{sing(m)}} \tag{4.7}$$

$$\langle\langle \text{ smile, j; 1}\rangle\rangle \vee \langle\langle \text{ sing, m; 1}\rangle\rangle \tag{4.8}$$

4.2.3 Propositions

Infons are just items of information and can neither be true or false in and by themselves. For an infon to be true or false, it must be an item of information about some *situation*. The observation is central to Situation Semantics, and it is important to distinguish infons related to situations and those unrelated. We call the infon related to situations *propositions*. Suppose from a situation s one extracts an item of information such as *"John smiles"*, $\langle\langle$ smile, j; 1$\rangle\rangle$, possibly by seeing the situation. The proposition that John smiles at the situation s is expressed as either (4.9) or (4.10). The proposition is *true* if he certainly smiles at the situation, *false* otherwise.

$$\boxed{\begin{array}{l}\boxed{\text{s}}\\ \quad\text{smile(j)}\end{array}} \tag{4.9}$$

$$(s \models \langle\langle \text{ smile, j; 1}\rangle\rangle) \tag{4.10}$$

We say that a situation s *supports* an infon σ, or that a situation s is of *type* σ when the proposition is true. We can combine propositions with the connectives, $\wedge$ and $\vee$, to construct more complex propositions as is the case with infons.

4.2.4 Parametric objects

In the above, we only consider infons whose arguments are some concrete objects, e.g., John. There can however be infons some of whose arguments are to be filled with some objects. Suppose we think of an abstract idea of *"smile"*, and it does not matter whoever smiles actually. Then, it is appropriate to use a parameter, x, to fill the argument to express it can be anyone. We call such infons including parameters *parametric* infons. The abstract idea of *"smile"* may be expressed as below (4.11) or (4.12).

$$\boxed{\text{smile}(x)} \tag{4.11}$$

$$\langle\langle \text{ smile, } x\text{; 1}\rangle\rangle \tag{4.12}$$

However it does not make the theory complete just introducing a parameter to the object. We have to incorporate parameters in a theoretically sound manner. Traditional approach to parameters is to regard them as variables and express abstract ideas using the λ-abstraction. For example, the concept of *"smile"* may be expressed as $\lambda x.\text{smile}(x)$. Such an idea is exploited by the λ-DRT [BMM+94], and the example might be expressed in the theory as

$$\lambda x \boxed{\text{smile}(x)} \tag{4.13}$$

The approach taken by Situation Theory is, on the other hand, more radical than the λ-calculus. While in the calculus the order of abstractions does matter, it does not for the abstraction adopted in Situation Semantics, thus called *simultaneous* abstraction. For example, in the λ-calculus, the meanings of two expressions, $\lambda xy.[\text{eat}(x, y)]$ and $\lambda yx.[\text{eat}(x, y)]$, are different because the former is an abbreviation of $\lambda x \lambda y.[\text{eat}(x, y)]$, while the latter is that of $\lambda y \lambda x.[\text{eat}(x, y)]$, which are different functions in the λ-calculus.

4.2.5 Abstraction

To abstract semantic objects simultaneously over several parameters, we introduce to the theory a notion of *role*, which indicates which abstraction is to be applied to which assignment function. With these roles, we obtain more flexibility in abstracting semantic objects over parameters because we do not need to specify in what order these parameters should be substituted. Let us see some examples. When we abstract the object above with a role r, it is expressed as follows:

$$\begin{array}{|c|}\hline r \to x \\ \hline \text{smile}(x) \\ \hline \end{array} \tag{4.14}$$

$$\lambda[r \to x]\langle\!\langle\, \text{smile},\ x;\ 1 \rangle\!\rangle \tag{4.15}$$

With this notation, we express the abstract idea on eating relation as (4.16) or (4.17):

$$\begin{array}{|c|}\hline r_1 \to x,\ \ r_2 \to y \\ \hline \text{eat}(x,\ y) \\ \hline \end{array} \tag{4.16}$$

$$\lambda[r_1 \to x, r_2 \to y]\langle\!\langle\, \text{eat},\ x,\ y;\ 1 \rangle\!\rangle \tag{4.17}$$

Since the roles serve as a key to get access to a particular parameter, the order of abstractions over x and y does not matter. We can swap x and y as follows without changing the meaning:

$$\begin{array}{|c|}\hline r_2 \to y,\ \ r_1 \to x \\ \hline \text{eat}(x,\ y) \\ \hline \end{array} \tag{4.18}$$

$$\lambda[r_2 \to y, r_1 \to x]\langle\!\langle\, \text{eat},\ x,\ y;\ 1 \rangle\!\rangle \tag{4.19}$$

In addition, we assume α-equivalence between abstract objects, i.e., alphabetical variances for parameters do not matter. Also, not only basic infons but also propositions can be abstracted in the same way. If we abstract the proposition (4.9) or (4.10) over the parameter x, which replaced 'j', we get the proposition abstracts, (4.20) or (4.21).

$$\begin{array}{|c|}\hline r \to x \\ \hline \boxed{s} \\ \text{smile}(x) \\ \hline \end{array} \tag{4.20}$$

$$\lambda[r \to x](s \models \langle\!\langle\, \text{smile},\ x;\ 1 \rangle\!\rangle) \tag{4.21}$$

4.2.6 Anchoring

Abstract objects are turned into concrete objects, i.e., basic infons or propositions, when assigned objects to parameters. Suppose we assign 'j', the person John, to x in the infon abstracts shown as (4.14) or (4.15). Such operation is expressed either as (4.22) or (4.23):

$$\begin{array}{|c|}\hline r \to x \\ \hline \text{smile}(x) \\ \hline \end{array}\;[r \to \text{j}] \tag{4.22}$$

$$\lambda[r \to x]\langle\!\langle\, \text{smile},\, x;\, 1 \rangle\!\rangle[r \to \text{j}] \tag{4.23}$$

As the result of the application, we obtain the basic infon (4.1) or (4.2), which means that *"John smiles."*:

$$\boxed{\text{smile(j)}} \tag{4.1}$$

$$\langle\!\langle\, \text{smile, j; 1} \rangle\!\rangle \tag{4.2}$$

The same goes for proposition abstracts and propositions. In Situation Semantics, the abstract object provided with assignments and its result of application are thought to be identical, i.e., β-conversion.

4.3 A study of primitive Situation Semantics

We represent semantics objects employed in primitive Situation Semantics in C-$\mathbf{ILL}_e^=$, the intuitionistic combinatorial linear logic with equality. We encode them first into the π-calculus or $\mathcal{L}_0$ if necessary, then translate them to the formulas in the logic.

4.3.1 Basic infons

We regard infons as processes providing for items of information and conceive of them as corresponding to output processes. In encoding them into the π-calculus we take relations as channels and arguments as data emitted through them. This is the approach taken by Fujinami [Fuj94] and Okada [Oka93]. For example, the infon, $\langle\!\langle$ smile, j; 1$\rangle\!\rangle$, shown as (4.1) or (4.2), is encoded as

$$\overline{smile}\langle \text{j} \rangle \tag{4.24}$$

The encoding may appear to be strange, but remember how actions are usually coded in *prolog*. Propositions such as *"Mary sings"* or *"John eats an apple"* are for example often coded as `sing(mary)` or `eat(john, apple)`, respectively, where one translates actions to relations. Comparing with the practice in *prolog* programming, representing actions as channels is rather natural. One may even admit to translate relations or properties to channels, then.

When we encode infons into the π-calculus, we do not express the polarity, 1, because we relate *capability* of performing actions to '1' and *incapability* to '0'. The point is that one does not define incapable actions in the π-calculus because it is designed to define capable actions only. That is why we can only express positive infons in the π-calculus. But it can be expressed in the logic. We translate it to the formula (4.25) [1] and the negative infon $\langle\!\langle$ smile, j; 0$\rangle\!\rangle$, shown in (4.4), to the formula (4.26).

$$\overline{smile}\langle \text{j} \rangle \tag{4.25}$$

$$\overline{smile}\langle \text{j} \rangle \multimap \mathbf{0} \tag{4.26}$$

The representation however contradicts our intuition that the infon such as *"John smiles"* is available for the time being. At least, it should not disappear just after providing for the information once. Given the observation, the infon had better be encoded in the π-calculus as a replicable process, e.g., $!\overline{smile}\langle \text{j} \rangle$. But we cannot translate replicable processes to formulas in the logic. We therefore give up with the intuition and regard the investigation to be done at more abstract level.

4.3.2 Conjunctive and disjunctive infons

To encode conjunctive infons into the π-calculus, we use the choice operator, $+$. For example, the conjunctive infon, $\langle\!\langle$ smile, j; 1$\rangle\!\rangle \wedge \langle\!\langle$ sing, m; 1$\rangle\!\rangle$, shown as (4.5) or (4.6), is encoded as

$$\overline{smile}\langle \text{j} \rangle + \overline{sing}\langle \text{m} \rangle \tag{4.27}$$

We relate the choice operator to additive conjunction, &, and represent it as

[1]The expression is an abbreviation for $export(smile, \text{j}) \Rightarrow \mathbf{1}$.

$$\overline{smile}\langle \mathrm{j}\rangle \;\&\; \overline{sing}\langle \mathrm{m}\rangle \tag{4.28}$$

The formula may be constructed as product of two formulas below, (4.29) and (4.30). The formula (4.31) states that $\overline{smile}\langle \mathrm{j}\rangle$ and $\overline{sing}\langle \mathrm{m}\rangle$ can be executed under any condition. Remember $\overline{smile}\langle \mathrm{j}\rangle + \overline{sing}\langle \mathrm{m}\rangle$ can be regarded as $\mathbf{1}.(\overline{smile}\langle \mathrm{j}\rangle + \overline{sing}\langle \mathrm{m}\rangle)$.

$$\mathbf{1} \Rightarrow \overline{smile}\langle \mathrm{j}\rangle \tag{4.29}$$
$$\mathbf{1} \Rightarrow \overline{sing}\langle \mathrm{m}\rangle \tag{4.30}$$
$$\mathbf{1} \Rightarrow \overline{smile}\langle \mathrm{j}\rangle \;\&\; \overline{sing}\langle \mathrm{m}\rangle \tag{4.31}$$

Disjunctive infons cannot be encoded into the π-calculus since it is not equipped with an operator corresponding to disjunction. We can however express them in the logic with additive disjunction, $\oplus$. For example, a disjunctive infon, $\langle\!\langle$ smile, j; 1$\rangle\!\rangle \vee \langle\!\langle$ sing, m; 1$\rangle\!\rangle$, shown as (4.7) or (4.8), can be defined as $\overline{smile}\langle \mathrm{j}\rangle \oplus \overline{sing}\langle \mathrm{m}\rangle$, constructed as coproducts of the formulas below, (4.32) and (4.33).

$$\overline{smile}\langle \mathrm{j}\rangle \Rightarrow \mathbf{1} \tag{4.32}$$
$$\overline{sing}\langle \mathrm{m}\rangle \Rightarrow \mathbf{1} \tag{4.33}$$
$$\overline{smile}\langle \mathrm{j}\rangle \oplus \overline{sing}\langle \mathrm{m}\rangle \Rightarrow \mathbf{1} \tag{4.34}$$

4.3.3 Links

Situations and propositions

For convenience, we first regard the notion of *situations* in Situation Semantics to correspond to processes. The relation between situations and infons such as $(s \models \sigma)$, a situation s supports an infon σ, can be regarded to correspond to the relation between terms and formulas, by relating terms to situations and formulas to infons. Then the proposition, $(s \models \langle\!\langle$ smile, j; 1$\rangle\!\rangle)$, shown as (4.9) or (4.10), is thought to represent the following expression in the logic:

$$s : \mathbf{1} \Rightarrow \overline{smile}\langle \mathrm{j}\rangle \tag{4.35}$$

If we define it as an axiom, then it is true. An expression can also be true if it is constructed of the set of given axioms. For example, the expression (4.38) is true when

the axioms, (4.36) and (4.37), are given because it can be constructed as product of them.

$$s : \mathbf{1} \Rightarrow \overline{smile}\langle \mathrm{j}\rangle \tag{4.36}$$
$$t : \mathbf{1} \Rightarrow \overline{sing}\langle \mathrm{m}\rangle \tag{4.37}$$
$$\langle s, t\rangle : \mathbf{1} \Rightarrow \overline{smile}\langle \mathrm{j}\rangle \;\&\; \overline{sing}\langle \mathrm{m}\rangle \tag{4.38}$$

The logic can thus determine whether an expression is *true* relative to given axioms if it is the case. If it can be constructed with given axioms, then it is true. Given the correspondence between terms and situations, the logic can be said to construct situations supporting infons. Unfortunately, the logic cannot determine whether or not a given expression is *false* because the logic is undecidable [Tro92].

Connections and links

To elaborate the above notion of situations, we turn to another notion of *connections* developed in Channel Theory. Observe that what we represent with $\Rightarrow$ corresponds to *constraints* between infons. For example, the following formula (4.39) states a constraint such that *"If John smiles and Mary sings, then John smiles"* rather than an infon supported by a particular situation.

$$\overline{smile}\langle \mathrm{j}\rangle \;\&\; \overline{sing}\langle \mathrm{m}\rangle \Rightarrow \overline{smile}\langle \mathrm{j}\rangle \tag{4.39}$$

Connections are the counterpart of situations and can be thought to support constraints in the same sense that situations support infons. The relation between these two sets of notions can be depicted as is shown in the figure 4.1 and explained as follows:

- A particular situation s supports an infon σ, *"John smiles and Mary sings"*, which we express as a proposition, $(s \models \sigma)$.
- Another particular situation t supports an infon ς, *"John smiles"*, which we express as a proposition, $(t \models \varsigma)$.
- There is a *connection*, c, between the situations s and t, which we express as $s \overset{c}{\mapsto} t$.
- There is a *constraint*, γ, between the infons σ and ς, which we express as $\sigma \overset{\gamma}{\Longrightarrow} \varsigma$.
- The particular connection c supports the constraint γ, which we express as a *link*, $(c \models \gamma)$. The term, *link*, is introduced as a counterpart of *proposition*.

With this conceptualization, the following expression (4.40) whose term is u can be thought to represent a link, where u is regarded as a connection.

$$u : \overline{smile}\langle \mathrm{j}\rangle \;\&\; \overline{sing}\langle \mathrm{m}\rangle \Rightarrow \overline{smile}\langle \mathrm{j}\rangle \tag{4.40}$$

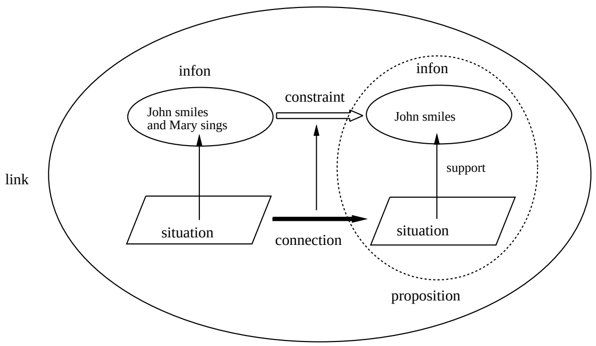


Figure 4.1. Constraints, connections, and links

The situation supporting the antecedent, *"John smiles and Mary sings"*, is the precondition of u, and another situation supporting the consequence, *"John smiles"*, is the postcondition of u. Let $^\bullet u$ and $u^\bullet$ be the precondition and postcondition of u, respectively. Then, the above expression (4.40) can be transformed into a link as below:[2]

$$\begin{array}{ccc} \overline{smile}\langle \mathrm{j}\rangle \ \& \ \overline{sing}\langle \mathrm{m}\rangle & \stackrel{\gamma}{\Longrightarrow} & \overline{smile}\langle \mathrm{j}\rangle \\ \models & \models & \models \\ ^\bullet u & \stackrel{u}{\mapsto} & u^\bullet \end{array}$$

Therefore, what the logic actually constructs through derivation are connections supporting constraints, not situations supporting infons.

4.3.4 Parametric infons

We have to recover the distinction between *constants* and *variables* to encode parametric infons in the π-calculus. Such modification is not problematic at all because the notion of *names* was originally introduced to simplify the calculus by discarding the distinction [MPW92].[3] Variables are special channels through which no names are exchanged. Constants are names such that no other names can substitute them. They can also not be used as channels. Hereinafter, we let $a, b, \ldots$, range over constants, and $x, y, \ldots$,

[2] The pre and post conditions are identical in logically valid inference, but they may be different for constraints in general.

[3] The Pict language for example distinguishes variables and constants, a language whose design is based on the π-calculus[Pie94].

variables. With this modification, the parametric infon, $\langle\!\langle$ smile, x; 1$\rangle\!\rangle$, shown as (4.11) or (4.12), can be encoded as

$$\overline{smile}\langle x\rangle \tag{4.41}$$

It is trivial to translate the process into the logic since it distinguishes constants and variables. The above x would be treated as a free variable.

4.3.5 Abstracts

Infon abstracts

The attempt to encode situation-theoretic objects into the π-calculus turns out to be most beneficial when we encode abstract objects. As we have seen in the first chapter (§1.5.2), the calculus enables us to encode simultaneous abstractions naturally. In the following process, P, for example, the guard $r(x)$ behaves like the λ-prefix, λx, in that it binds x in $\overline{smile}\langle x\rangle$.

$$P =_{\mathrm{def}} r(x).\overline{smile}\langle x\rangle$$

In this guise, the process certainly encodes the infon abstract, $\lambda[r \to x]\langle\!\langle$ smile, x; 1$\rangle\!\rangle$, shown as (4.14) or (4.15), where the channel r corresponds to the role indexing the parameter x.

The encoding of abstracts as input actions seems to work fine, but there is a problem, for which we employ molecular language. Consider how the infon abstract, $\lambda[r_1 \to x, r_2 \to y]\langle\!\langle$ eat, x, y; 1$\rangle\!\rangle$, shown as(4.16) or (4.17), can be encoded in the π-calculus. We may want to encode it as either

$(\ r_1(x) \ \mid r_2(y) \ ). \ \overline{eat}\langle x, y\rangle$, or

$(\ r_2(y) \ \mid r_1(x) \ ). \ \overline{eat}\langle x, y\rangle$

The encoding expresses our intuition on simultaneous abstractions; the process imports two constants through r_1 and r_2 simultaneously and substitutes them for x and y, respectively. But the problem is that the syntax of the calculus does not allow parallel processes to guard an action. We can only encode it either as

$r_1(x).r_2(y).\overline{eat}\langle x, y\rangle$, or

$r_2(y).r_1(x).\overline{eat}\langle x, y\rangle$

and as the result the abstraction looks more like the λ-abstraction. Making them independent processes as below does not work because x and y in the first two processes are not the same as those appearing in $\overline{eat}\langle x, y\rangle$.

$$r_1(x) \;|\; r_2(y) \;|\; \overline{eat}\langle x, y\rangle$$

This is because the scope of names bound by these input actions are limited within the guarded actions by them, which does not apply to the case. Another way to use the restriction operator, too, cannot extend the scope of x and y across $|$ because the binding forces by the input actions are stronger than that of the restriction operator. The following two definitions are therefore equivalent.

$$(\nu\, x, y)(r_1(x) \;|\; r_2(y) \;|\; \overline{eat}\langle x, y\rangle)$$
$$r_1(x) \;|\; r_2(y) \;|\; (\nu\, x, y)(\overline{eat}\langle x, y\rangle)$$

The molecular language $\mathcal{L}_0$, on the other hand, is less restrictive with respect to the scope of binding. Since the scope of bound names is defined to be within the molecule in which they appear, we can encode it as below.

$$[\, r_1(x), r_2(y), \overline{eat}\langle x, y\rangle \,] \tag{4.42}$$

Now consider how the above program can be translated into the logic. Unfortunately, it is not straightforward because we have to compile bound names to unique constants. If we translated it without compiling them, we would end up with the following set of axioms, (4.43) to (4.45). The problem is that the scope of variables cannot go across between axioms.

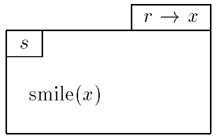
$$s : r_1(x) \Rightarrow \mathbf{1} \tag{4.43}$$
$$t : r_2(y) \Rightarrow \mathbf{1} \tag{4.44}$$
$$u : \overline{eat}\langle x, y\rangle \Rightarrow \mathbf{1} \tag{4.45}$$

Let $\mathtt{x}$ and $\mathtt{y}$ be unique constants assigned to x and y in (4.42), respectively, and rewrite the above axioms to those below. Since they are constants, they can now denote the same object among different axioms.

$$s : r_1(\mathtt{x}) \Rightarrow \mathbf{1} \tag{4.46}$$
$$t : r_2(\mathtt{y}) \Rightarrow \mathbf{1} \tag{4.47}$$
$$u : \overline{eat}\langle \mathtt{x}, \mathtt{y}\rangle \Rightarrow \mathbf{1} \tag{4.48}$$

Proposition abstracts

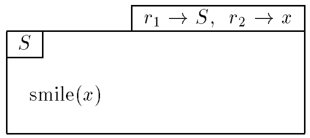
The above translation works for infon abstracts but not for proposition abstracts, the abstract objects obtained by abstracting propositions over parameters. Recall the abstract object obtained of the parametric proposition, $(s \models \langle\!\langle$ smile, x; $1\rangle\!\rangle)$, by abstracting it over x was the one such as (4.20) or (4.21), not (4.49).



(4.20)

$$\lambda[r \to x](s \models \langle\!\langle \text{ smile, } x;\, 1\rangle\!\rangle) \tag{4.21}$$
$$(s \models \lambda[r \to x]\langle\!\langle \text{ smile, } x;\, 1\rangle\!\rangle) \tag{4.49}$$

The definition (4.21) means that the abstraction can operate on the situation s, too. In fact, Situation Semantics allows to parameterise the situation s and abstract the proposition over it as follows:



(4.50)

$$\lambda[r_1 \to S, r_2 \to x](S \models \langle\!\langle \text{ smile, } x;\, 1\rangle\!\rangle) \tag{4.51}$$

Since terms and formulas are not interchangeable, we can neither encode such abstract objects into the π-calculus nor translate it to the logic. This might appear to be problematic, but we will approach the problem from a different direction in §4.7.3, where we regard situations as channels to infons.

4.3.6 Assignments and transitions

Assignments are encoded as output actions connected through channels to input actions abstracting objects. Suppose we encode in the molecular language, $\mathcal{L}_0$, the infon abstract, $\lambda[r \to x]\langle\!\langle$ smile, x; $1\rangle\!\rangle$, shown as (4.14) or (4.15), as follows (4.52), where the channel r corresponds to the role indexing the parameter x.

$$[\, r(x), \overline{smile}\langle x\rangle \,] \tag{4.52}$$

Then, an assignment anchoring x to 'j' can be encoded as an output action, $\overline{r}\langle \mathrm{j}\rangle$. The abstract object with the assignment, $\lambda[r \to x]\langle\!\langle$ smile, x; 1$\rangle\!\rangle[r \to j]$ can, therefore, be encoded as the following molecule:

$$[\, \overline{r}\langle \mathrm{j}\rangle, r(x), \overline{smile}\langle x\rangle \,] \tag{4.53}$$

The molecule will evolve into $[\overline{smile}\langle \mathrm{j}\rangle]$ upon the interaction via the channel r as below. The end result corresponds to the infon, $\langle\!\langle$smile, j; 1$\rangle\!\rangle$.

$$\begin{array}{ll} & [\, \overline{r}\langle \mathrm{j}\rangle, r(x), \overline{smile}\langle x\rangle \,] \\ \stackrel{\tau}{\searrow} & [\, \overline{smile}\langle x\rangle \,] * \{^{\mathrm{j}}/_x\} \\ \equiv & [\, \overline{smile}\langle \mathrm{j}\rangle \,] \end{array}$$

When translated to the logic, the molecule is expressed as the set of the following axioms:

$$\begin{array}{l} s:\ r_1(\mathtt{x}) \Rightarrow \mathbf{1} \\ t:\ \overline{r_1}\langle \mathtt{j}\rangle \Rightarrow \mathbf{1} \\ u:\ \overline{smile}\langle \mathtt{x}\rangle \Rightarrow \mathbf{1} \\ v:\ r_1(\mathtt{x}) \otimes \overline{r_1}\langle \mathtt{j}\rangle \Rightarrow !(\mathtt{x} = \mathtt{j}) \\ id:\ \overline{smile}\langle \mathtt{x}\rangle \Rightarrow \overline{smile}\langle \mathtt{x}\rangle \end{array}$$

and the transition is derived as below:[4]

$$\dfrac{\dfrac{v:\ r_1(\mathtt{x}) \otimes \overline{r_1}\langle \mathtt{j}\rangle \Rightarrow !(\mathtt{x} = \mathtt{j}) \quad id:\ \overline{smile}\langle \mathtt{x}\rangle \Rightarrow \overline{smile}\langle \mathtt{x}\rangle}{v \otimes id : r_1(\mathtt{x}) \otimes \overline{r_1}\langle \mathtt{j}\rangle \otimes \overline{smile}\langle \mathtt{x}\rangle \Rightarrow !(\mathtt{x} = \mathtt{j}) \otimes \overline{smile}\langle \mathtt{x}\rangle} \quad \mathbf{eq}_3 : !(\mathtt{x} = \mathtt{j}) \otimes \overline{smile}\langle \mathtt{x}\rangle \Rightarrow \overline{smile}\langle \mathtt{j}\rangle}{(v \otimes id) \circ \mathbf{eq}_3 : r_1(\mathtt{x}) \otimes \overline{r_1}\langle \mathtt{j}\rangle \otimes \overline{smile}\langle \mathtt{x}\rangle \Rightarrow \overline{smile}\langle \mathtt{j}\rangle}$$

The above example shows that β-conversion is realized as transition in our formulation. One may however wonder how the equivalence between the following two objects can be captured as argued in Situation Semantics.

- $\lambda[r \to x]\langle\!\langle$ smile, x; 1$\rangle\!\rangle[r \to j]$, and
- $\langle\!\langle$ smile, j; 1$\rangle\!\rangle$

[4] We skip the steps to swap (x=j) to (j=x).

To capture the equivalence, we turn to the notion of *bisimular* relation, $\sim$. Intuitively, the relation $A \sim B$ means that a process A can simulate the behaviour of another process B and vice versa. Observe how the two processes, P and Q, encoding the above two objects may behave. In the table below, the lefthand side shows the behaviour of the process, P, encoding $\lambda[r \to x]\langle\!\langle$ smile, x; 1$\rangle\!\rangle[r \to j]$, and the righthand side that of the process, Q, encoding $\langle\!\langle$ smile, j; 1$\rangle\!\rangle$, as a molecule.

$$\begin{array}{ll||ll} & P & & Q \\ & [\, \overline{r}\langle \mathrm{j}\rangle, r(x), \overline{smile}\langle x\rangle \,] & & \\ \stackrel{\tau}{\searrow} & [\, \overline{smile}\langle x\rangle \,] * \{^{\mathrm{j}}/_x\} & & \\ \equiv & [\, \overline{smile}\langle \mathrm{j}\rangle \,] & & [\, \overline{smile}\langle \mathrm{j}\rangle \,] \\ \stackrel{\overline{smile}\langle \mathrm{j}\rangle}{\searrow} & \emptyset & \stackrel{\overline{smile}\langle \mathrm{j}\rangle}{\searrow} & \emptyset \end{array}$$

The left process, P, performs the sequence of actions, $\langle\ \tau, \overline{smile}\langle \mathrm{j}\rangle\ \rangle$, while the right one, Q, the single action, $\langle\ \overline{smile}\langle \mathrm{j}\rangle\ \rangle$. These two sequences may appear to be different, but recall that the silent action, τ, is invisible from outside. We can neglect τ and can regard these two as equivalent. We define *weak* bisimular relation, $\sim_w$, as the process equivalence neglecting the silent action, τ. Obviously, it holds that $P \sim_w Q$.[5] When we conceive of transitions as LTS, the issue of bisimulation is reduced to mapping between directed acyclic graphs. An interesting question for us is how dataflow graphs devised in the thesis may contribute to defining a bisimulation relation, but it is outside the scope of the thesis.

4.4 Naive Situation Theory

The version of situation theory presented so far has been slightly simplified in order to explain our ideas to investigating Situation Semantics. Some objects employed in Situation Semantics are in fact more complex. In this section, we look into *infons* with *appropriate conditions*, *relations*, and *types*. We call the theory *naive* because we still postpone the study *constraints* and *quantifications*.

4.4.1 Infons

As we have seen, an infon consists of three elements: relation, argument, and polarity. Of which, we elaborate the notion of *relation* now. Relations are objects in Situation Semantics and have a type of assignments associated with their appropriateness conditions. Relations are a particular kind of abstract and thus they have indexed roles and

[5] Strong bisimular relation, $\sim_s$, may be defined as the equivalence where the silent action, τ, counts. See Milner's text [Mil89] for more detailed explanation and definitions.

restrictions on their assignments. The type of assignments associated with a relation characterises what assignments are appropriate to it. This means that a more explicit and correct notation for infons should mention the assignments and indices explicitly [Coo92]. The basic infon, shown as (4.1) or (4.2), should for example be expressed more precisely as (4.54) or (4.55) with a role r:

$$\boxed{\text{smile(j)}} \tag{4.1}$$

$$\langle\!\langle \text{ smile, j; 1} \rangle\!\rangle \tag{4.2}$$

$$\boxed{\text{smile(}\ r \to \text{j)}} \tag{4.54}$$

$$\langle\!\langle \text{ smile, } r \to \text{j; 1} \rangle\!\rangle \tag{4.55}$$

To encode the infon into the π-calculus, we have to change our encoding since the straightforward translation (4.56) is not possible. This is because the calculus does not allow for processes to be exchanged through channels. Although such a restriction will be lifted in the higher-order π-calculus, we opt for the first order version to keep our theory as simple as we can.

$$\overline{smile}\langle\, \overline{r}\langle \text{j}\rangle\, \rangle \tag{4.56}$$

Our solution is to encode it as a system composed of two processes, (4.57).

$$(\nu\, r)(\overline{smile}\langle r\rangle \mid \overline{r}\langle \text{j}\rangle) \tag{4.57}$$

The system will emit the role r from the channel $smile$. Since the emitted role r, too, serves as a channel through which the object, 'j', can be emitted, some other process having access to the process, $\overline{smile}\langle r\rangle$, can successfully get access to the object. In this encoding, the assignment, $r \to$ j, to the relation, smile, is encoded in the same way as that of abstract objects such as infon abstracts or proposition abstracts. The encoding into $\mathcal{L}_0$ and translation into the logic can thus be done in the same manner. The system (4.57) can be expressed in $\mathcal{L}_0$ as (4.58).

$$(\nu\, r)[\, \overline{smile}\langle r\rangle, \overline{r}\langle \text{j}\rangle\,] \tag{4.58}$$

4.4.2 Abstract objects and assignments

As we have changed our encoding of infons, we have to modify our encoding of abstract objects. For example, how can the following infon abstract, (4.14) or (4.15), be encoded?

$$\begin{array}{r} \boxed{r \to x} \\ \boxed{\text{smile}(x)} \end{array} \tag{4.14}$$

$$\lambda[r \to x]\langle\!\langle \text{ smile, } x\text{; 1} \rangle\!\rangle \tag{4.15}$$

Our solution is not so different from the original encoding. We replace r for x in $\overline{smile}\langle x\rangle$ as (4.59).

$$(\nu\, r)[\, \overline{smile}\langle r\rangle, r(x)\,] \tag{4.59}$$

The encoding of assignments is however different from the previous. To understand the motivation, we look into the problem which we encounter if we would stick to the previous encoding. Assume we add a particle encoding an assignment, e.g., $\overline{r}\langle \text{j}\rangle$, to the molecule (4.59). Then, it would evolve as follows:

$$\begin{array}{ll} & (\nu\, r)[\, \overline{smile}\langle r\rangle, r(x), \overline{r}\langle \text{j}\rangle\,] \\ \searrow_{\tau} & (\nu\, r)[\, \overline{smile}\langle r\rangle\,]*\{^{\text{j}}/_{x}\} \end{array}$$

Although the variable x has certainly been substituted by 'j', not only the input process, $r(x)$, but also the output process, $\overline{r}\langle \text{j}\rangle$, encoding the assignment, have disappeared. To avoid the trouble, we encode the assignment as a replicable process, i.e., $!\,\overline{r}\langle \text{j}\rangle$. The encoding fits with our intuition that sources of information which we plug in through a channel may provide us with items of information as many times as we request for. To implement the idea, we equip the language with replication, !.[6] Given the modification,

[6] See Appendix A for definition.

the encoding and transition becomes as follows:[7]

$$\begin{array}{ll} & (\nu\, r)[\,\overline{smile}\langle r\rangle, r(x), !\,[\,\overline{r}\langle \mathrm{j}\rangle\,]\,] \\ \equiv & (\nu\, r)[\,\overline{smile}\langle r\rangle, r(x), \overline{r}\langle \mathrm{j}\rangle, !\,[\,\overline{r}\langle \mathrm{j}\rangle\,]\,] \\ \overset{}{\underset{\tau}{\searrow}} & (\nu\, r)[\,\overline{smile}\langle r\rangle, !\,[\,\overline{r}\langle \mathrm{j}\rangle\,]\,]*\{{}^{\mathrm{j}}/_{x}\} \end{array}$$

Observing the encoding and transition, one may wonder what use of the input process, $r(x)$, actually is. It is in fact possible to exclude it and take simply $\overline{smile}\langle r\rangle$ as an infon abstract. That is, the infon abstract shown as (4.14) or (4.15) can be encoded as (4.60). To discuss the issue, we have to investigate another sort of complex object, *relation*.

$$(\nu\, r)[\,\overline{smile}\langle r\rangle\,] \tag{4.60}$$

4.4.3 Relations

In Situation Semantics, there is an idea called *predication*. To understand the idea, observe that the infon abstracts shown as (4.14) and (4.15) can be regarded as a relation, whose argument must be filled in with an assignment. To distinguish infon abstracts serving as relations from those mere abstract objects, we express infon abstracts as relations as (4.61).

$$\begin{array}{|c|}\hline r \to x \\ \hline \text{smile}(x) \\ \hline \end{array} \tag{4.61}$$

Since it is a relation, it can constitute an infon given an assignment such as (4.62). When we compare it with the basic infon (4.54), the difference is subtle.

[7] It is not desirable to introduce replication because replicable processes cannot be dealt within the logic. The problem will partly be tempered in next subsection.

$$\boxed{\begin{array}{|c|}\hline r \to x \\ \hline \text{smile}(x) \\ \hline \end{array}\ (\mathrm{r} \to \mathrm{j})} \tag{4.62}$$

$$\boxed{\text{smile}(\ r \to \mathrm{j})} \tag{4.54}$$

It is argued that they are equivalent in the sense that a situation supports (4.62) if and only if it supports (4.54). They are however different in structure. Such a difference is utilised in Situation Theoretic Grammar, which is one of the reasons to maintain the difference [Coo92]. For example, utterances of words and phrases are used as role indices on the relation. We will however take a different approach to grammar and therefore do not see particular reasons to maintain the difference. We rather regard them as identical objects so that we can assign to them the same term. We express thus both of them as is shown in (4.63). In the molecule, the assignment, $\overline{r}\langle \mathrm{j}\rangle$, need not be replicable because it will not be consumed by an input particle.

$$(\nu\, r)[\,\overline{smile}\langle r\rangle, \overline{r}\langle \mathrm{j}\rangle\,] \tag{4.63}$$

The decision that we regard both objects as identical further motivates us to regard both relations in (4.62) and (4.54) as identical object. We will hence regard the object (4.61) to be encoded as (4.60), where the variable x is excluded. It might make our intention clear to depict (4.63) in dataflow graph. Figure 4.2 shows the graph depicting the structure of the infon.

When we parameterise the object with respect to the first argument and abstract it over the parameter, the figure changes to that shown in the figure 4.3. From the difference, one can see that we represent infon abstracts as structured objects whose parts are missing, the parts that provide for items of information on the filler of argument.

4.4.4 Types

In the way that infon abstracts are turned into relations, proposition abstracts are turned into *types*. The proposition abstracts shown as (4.20) can for example be turned into a type such as shown in (4.64).

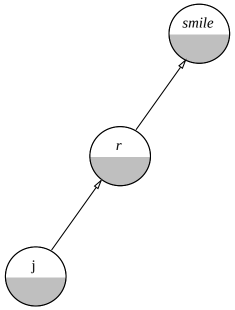


Figure 4.2. The structure of the infon, $\langle\!\langle$ smile, $r \to$ j; 1$\rangle\!\rangle$

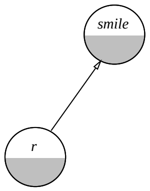
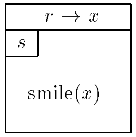
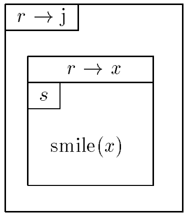


Figure 4.3. The structure of the infon abstract, $\lambda[r \to x]$ $\langle\!\langle$ smile, $r \to x$; 1$\rangle\!\rangle$

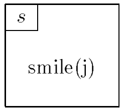


(4.64)

Then, what is typed by the type is its assignments. If we have an assignment such as [$r \to$ j], they would comprise a proposition such as (4.65), which is claimed to be equivalent, but not identical, to the object (4.66):



(4.65)



(4.66)

It takes us into a trouble to admit assignments to be typed in this way. Since our system admits only proofs as typeable, it conflicts with our principle to regard formulas specifying assignments as terms. We could for example define two axioms, $t : \overline{smile}\langle r\rangle \Rightarrow \mathbf{1}$ and $u : \overline{r}\langle \mathrm{j}\rangle \Rightarrow \mathbf{1}$, and construct the situation supporting the infon as below, which by no means corresponds to (4.65).

$$\frac{t : \overline{smile}\langle r\rangle \Rightarrow \mathbf{1} \quad u : \overline{r}\langle \mathrm{j}\rangle \Rightarrow \mathbf{1}}{t \otimes u : \overline{smile}\langle r\rangle \otimes \overline{r}\langle \mathrm{j}\rangle \Rightarrow \mathbf{1}}$$

The use of types was partly motivated to analyse propositional attitudes such as beliefs [Coo92]. Although we cannot define situation-theoretic types in the logic, we approach the problem from a different perspective in §4.7.3.

4.5 Constraints

The issue of constraints is still open to debate in Situation Semantics. We exploit therefore our own ideas to investigate it. To express constraints which always hold, we introduce to our logic a 'necessary' operator, $\Box$. We also introduce a 'possible' operator, $\Diamond$, to express constraints which sometimes hold. These operators will be elaborated in the next section (§4.6) to deal with quantification, where $\Box$ is related to $\forall$, and $\Diamond$ to $\exists$. For simplicity, we are back in this section to the primitive Situation Theory (§4.2) when encoding basic infons.

4.5.1 Constraints as transitions

As explained in §4.3.3, a constraint, $\phi \stackrel{\gamma}{\Longrightarrow} \psi$, is captured in terms of state change in our approach. We apply the idea of constraint to studying the meaning of conditional sentences such as (4.67) and represent it as a process such that an event where *"Mary sings"* is followed by another where *"John smiles."*

(4.67) If Mary sings, then John smiles.

If we encode such a process into the π-calculus, it may be encoded as

$$\overline{sing}\langle \mathrm{m}\rangle.\overline{smile}\langle \mathrm{j}\rangle.\mathbf{0}$$

which may be specified in our logic by the following expression:

$$s : \overline{sing}\langle \mathrm{m}\rangle \Rightarrow \overline{smile}\langle \mathrm{j}\rangle$$

4.5.2 Necessity

The approach can represent the meaning of a particular constraint, but how can we extend it to studying the meaning of more general constraint such as (4.68)?

(4.68) Whenever Mary sings, John smiles.

When encoded as a process, it may be expressed as (4.69), which behave as $\overline{sing}\langle \mathrm{m}\rangle.\overline{smile}\langle \mathrm{j}\rangle.\mathbf{0}$ no matter which one is chosen. The way to represent the meaning of universally quantified sentences by means of non-deterministic choice is reminiscent to the approach to 'donkey' anaphora in original DRT [Kam84]. The other process (4.70) does on the other hand not represent the meaning because the third case corresponds to an event where an event of Mary's singing is followed by John's laugh.

$$\overline{sing}\langle \mathrm{m}\rangle.\overline{smile}\langle \mathrm{j}\rangle.\mathbf{0} + \overline{sing}\langle \mathrm{m}\rangle.\overline{smile}\langle \mathrm{j}\rangle.\mathbf{0} + \overline{sing}\langle \mathrm{m}\rangle.\overline{smile}\langle \mathrm{j}\rangle.\mathbf{0} \tag{4.69}$$

$$\overline{sing}\langle \mathrm{m}\rangle.\overline{smile}\langle \mathrm{j}\rangle.\mathbf{0} + \overline{sing}\langle \mathrm{m}\rangle.\overline{smile}\langle \mathrm{j}\rangle.\mathbf{0} + \overline{sing}\langle \mathrm{m}\rangle.\overline{laugh}\langle \mathrm{j}\rangle.\mathbf{0} \tag{4.70}$$

To represent the necessary constraint, we introduce to the logic an operator, $\bigwith$(big sum), a generalised operator of additive conjunction, &. Suppose the cases in (4.69) are translated to the following set of axioms:

$$s : \overline{sing}\langle \mathrm{m}\rangle \Rightarrow \overline{smile}\langle \mathrm{j}\rangle$$
$$t : \overline{sing}\langle \mathrm{m}\rangle \Rightarrow \overline{smile}\langle \mathrm{j}\rangle$$
$$u : \overline{sing}\langle \mathrm{m}\rangle \Rightarrow \overline{smile}\langle \mathrm{j}\rangle$$

Then, by conjoining them together, we get a formula, $\overline{sing}\langle \mathrm{m}\rangle \Rightarrow \bigwith\overline{smile}\langle \mathrm{j}\rangle$:[8]

$$\frac{s : \overline{sing}\langle \mathrm{m}\rangle \Rightarrow \overline{smile}\langle \mathrm{j}\rangle \quad t : \overline{sing}\langle \mathrm{m}\rangle \Rightarrow \overline{smile}\langle \mathrm{j}\rangle \quad u : \overline{sing}\langle \mathrm{m}\rangle \Rightarrow \overline{smile}\langle \mathrm{j}\rangle}{\langle s,t,u\rangle : \overline{sing}\langle \mathrm{m}\rangle \Rightarrow \bigwith\overline{smile}\langle \mathrm{j}\rangle}\ (\&)$$

The meaning of $\bigwith$ in $\bigwith\overline{smile}\langle \mathrm{j}\rangle$ is reminiscent to the modal operator, $\Box$, in that $\overline{sing}\langle \mathrm{m}\rangle$ always leads to $\overline{smile}\langle \mathrm{j}\rangle$. We may therefore replace $\Box$ for $\bigwith$

$$\frac{s : \overline{sing}\langle \mathrm{m}\rangle \Rightarrow \overline{smile}\langle \mathrm{j}\rangle \quad t : \overline{sing}\langle \mathrm{m}\rangle \Rightarrow \overline{smile}\langle \mathrm{j}\rangle \quad u : \overline{sing}\langle \mathrm{m}\rangle \Rightarrow \overline{smile}\langle \mathrm{j}\rangle}{\langle s,t,u\rangle : \overline{sing}\langle \mathrm{m}\rangle \Rightarrow \Box\,\overline{smile}\langle \mathrm{j}\rangle}\ (\Box)$$

Since terms can be regarded as comprising a Kripke model, the following derivation should also be admissible:

$$\frac{v : \overline{sing}\langle \mathrm{m}\rangle \Rightarrow \overline{laugh}\langle \mathrm{j}\rangle}{\langle v\rangle : \overline{sing}\langle \mathrm{m}\rangle \Rightarrow \Box\,\overline{laugh}\langle \mathrm{j}\rangle}\ (\Box)$$

Given this observation, the above process (4.70) corresponds to the formula, $\overline{sing}\langle \mathrm{m}\rangle \Rightarrow \Box\,\overline{smile}\langle \mathrm{j}\rangle \;\&\; \Box\,\overline{laugh}\langle \mathrm{j}\rangle$, which is derived as below.

$$\frac{\dfrac{s : \overline{sing}\langle \mathrm{m}\rangle \Rightarrow \overline{smile}\langle \mathrm{j}\rangle \quad t : \overline{sing}\langle \mathrm{m}\rangle \Rightarrow \overline{smile}\langle \mathrm{j}\rangle}{\langle s,t\rangle : \overline{sing}\langle \mathrm{m}\rangle \Rightarrow \Box\,\overline{smile}\langle \mathrm{j}\rangle}\ (\Box) \quad \dfrac{v : \overline{sing}\langle \mathrm{m}\rangle \Rightarrow \overline{laugh}\langle \mathrm{j}\rangle}{\langle v\rangle : \overline{sing}\langle \mathrm{m}\rangle \Rightarrow \Box\,\overline{laugh}\langle \mathrm{j}\rangle}\ (\Box)}{\langle s,t,\langle v\rangle\rangle : \overline{sing}\langle \mathrm{m}\rangle \Rightarrow \Box\,\overline{smile}\langle \mathrm{j}\rangle \;\&\; \Box\,\overline{laugh}\langle \mathrm{j}\rangle}\ (\&)$$

4.5.3 Necessity in Hennessy-Milner Logic

It may clarify our approach to modality to compare our treatment with HML. We depict the processes, (4.69) and (4.70), as labelled transition systems to make the meaning explicit. The first process can be depicted as is shown in the figure 4.4, while the second one as is shown in the figure 4.5.

In Hennessy-Milner Logic, the first process (Figure 4.4) satisfies the formula (4.71) below but not (4.72). On the other hand, the second process (Figure 4.5) can satisfy both of them because HML formulas specify processes partially.

[8] We write $\langle s,t,v\rangle$ for $\langle\langle s,t\rangle,v\rangle$.

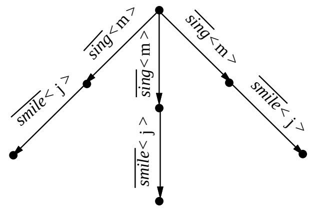


Figure 4.4. The case where John must smile when Mary sings

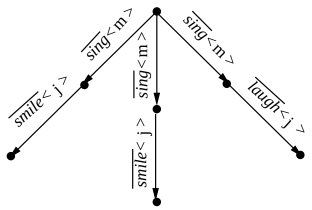


Figure 4.5. The case where John may laugh when Mary sings

$$[\overline{sing}\langle \mathrm{m}\rangle][\overline{smile}\langle \mathrm{j}\rangle]\mathbf{true} \tag{4.71}$$

$$[\overline{sing}\langle \mathrm{m}\rangle][\overline{smile}\langle \mathrm{j}\rangle]\mathbf{true} \wedge [\overline{sing}\langle \mathrm{m}\rangle][\overline{laugh}\langle \mathrm{j}\rangle]\mathbf{true} \tag{4.72}$$

The formula (4.71) cannot therefore specify the set of processes representing the meaning of *"Whenever Mary sings, John smiles."* To express such a property, one has to explicitly state that *"unacceptable, otherwise."* An extension of HML in that direction has been done by Stirling [Sti92] by introducing fixed-point operators. We do not, on the other hand, need to be bothered with the problem because we explicitly record terms satisfying the properties. That is, the formula, $\langle s, t, v\rangle : \overline{sing}\langle \mathrm{m}\rangle \Rightarrow \Box\, \overline{smile}\langle \mathrm{j}\rangle$, simply cannot be constructed from the following set of axioms:

$$s : \overline{sing}\langle \mathrm{m}\rangle \Rightarrow \overline{smile}\langle \mathrm{j}\rangle$$

$$t : \overline{sing}\langle \mathrm{m}\rangle \Rightarrow \overline{smile}\langle \mathrm{j}\rangle$$

$$v : \overline{sing}\langle \mathrm{m}\rangle \Rightarrow \overline{laugh}\langle \mathrm{j}\rangle$$

4.5.4 The axioms and rules for necessity and possibility

The axioms and rules for necessity, $\Box$, and possibility, $\Diamond$, are given in the table 4.2. These are analogous to those for products and coproducts (Table 4.3). We regard the big plus, $\bigoplus$, as $\Diamond$ as we regard the big sum, $\&$, as $\Box$. We therefore make terms reminiscent to those of products and coproducts. The term, $\langle \phi_1, \phi_2, \cdots, \phi_n\rangle$, gathers all the cases where A is always followed by B. The term π_i picks up a particular case from the cases satisfying A. The term, $[\phi_1, \phi_2, \cdots, \phi_n]$, gathers all the cases leading to A. The term, κ_i, turns a particular case satisfying A into a general statement of A.

Necessity:

$$\frac{\phi_1 : A \Rightarrow B \quad \phi_2 : A \Rightarrow B \quad \cdots\cdots \quad \phi_n : A \Rightarrow B}{\langle \phi_1, \phi_2, \cdots, \phi_n\rangle : A \Rightarrow \Box B}\ (\Box) \qquad \pi_i : \Box A \Rightarrow A_i$$

Possibility:

$$\frac{\phi_1 : B \Rightarrow A \quad \phi_2 : B \Rightarrow A \quad \cdots\cdots \quad \phi_n : B \Rightarrow A}{[\phi_1, \phi_2, \cdots, \phi_n] : \Diamond B \Rightarrow A}\ (\Diamond) \qquad \kappa_i : A_i \Rightarrow \Diamond A$$

Table 4.2. Axioms and rules for necessity and possibility

Products:

$$\frac{\phi : A \Rightarrow B \quad \psi : A \Rightarrow C}{\langle \phi, \psi \rangle : A \Rightarrow B \& C} \qquad \pi_{A,B,i} : A_0 \& A_1 \Rightarrow A_i \ (i \in \{0,1\})$$

Coproducts:

$$\frac{\phi : B \Rightarrow A \quad \psi : C \Rightarrow A}{[\phi, \psi] : B \oplus C \Rightarrow A} \qquad \kappa_{A,B,i} : A_i \Rightarrow A_0 \oplus A_1 \ (i \in \{0,1\})$$

Table 4.3. Axioms and rules for products and coproducts in C-$\mathbf{ILL}_0$

4.6 Quantification

4.6.1 The axioms and rules for quantifiers

Universal and existential quantifiers are introduced to the logic in analogous manner to modal operators; We replace $\forall$ and $\exists$ for $\Box$ and $\Diamond$, respectively. The difference is that formulas can be parameterised and that we record the mappings from constants to variables, and vice versa. Figure 4.4 shows the axioms and rules for quantifiers. They are explained as follows:

- The first rule for universal quantifier serves as introduction rule and allows one to generalise particular formulas to a universally quantified formula by parameterising constants to the same variable. These expressions B must be identical when they are parameterised. The proof of the quantified formula is the product of all proofs, where the mapping from the constant to the variable is recorded for each formula. For example, $\phi_x^{a_1}$ means that the constant, a_1 is replaced by x.
- The second rule for universal quantifier serves as elimination rule and allows to instantiate a universally quantified formula to a particular formula by substituting a constant, a, for a variable, x. The proof of the quantified formula is a projection such that it chooses a particular substitution environment.
- The first rule for existential quantifier introduces the quantifier in lefthand side and serves as elimination rule. The antecedent of each formula must be identical when its constant is parameterised with a variable. The proof of the obtained formula is the coproduct of proofs, where the mapping from the constant to the variable is recorded for each formula.
- The second rule for existential quantifier serves as introduction rule and allows to replace a variable for a particular constant. The proof is a function recording the mapping.

The variable, x, must be *eigenvariable* as usual for the first rules of universal and existential quantifiers. The equations are defined analogously with those for $\&$ and $\oplus$,

Universal quantifier:

$$\frac{\phi_1 : A \Rightarrow B[a_1/x] \ \cdots\cdots \ \phi_n : A \Rightarrow B[a_n/x]}{\langle \phi_x^{a_1}, \cdots, \phi_x^{a_n} \rangle : A \Rightarrow \forall x B} \ x \notin fv(A) \qquad \pi_x^a : \forall x A \Rightarrow A[a/x]$$

Existential quantifier:

$$\frac{\phi_1 : B[a_1/x] \Rightarrow A \ \cdots\cdots \ \phi_n : B[a_n/x] \Rightarrow A}{[\phi_x^{a_1}, \cdots, \phi_x^{a_n}] : \exists x B \Rightarrow A} \ x \notin fv(A) \qquad \kappa_x^a : A[a/x] \Rightarrow \exists x\, A$$

Table 4.4. Axioms and rules for quantifiers

respectively, as is shown in the table 4.5.

$$\langle \cdots, \phi_x^a, \cdots \rangle \circ \pi_x^a = \phi \qquad \kappa_x^a \circ [\cdots, \phi_x^a, \cdots] = \phi$$

Table 4.5. Equations for quantifiers

The rules for quantifiers are the same as those proposed by Engberg and Winskel for linear intuitionistic logic [EW93]. As is the case for their system, $\forall$ and $\exists$-adjunction hold as well. $\forall$-adjunction means $\phi : A \Rightarrow B$ and $\langle \phi_x \rangle : A \Rightarrow \forall x.B$ can be bi-directionally constructed. If the latter is assumed, then the former can be derived as is shown in the left of the table 4.6 by using π_x. By the equation, the term, $\langle \phi_x \rangle \circ \pi_x$, is equivalent to ϕ. The opposite direction is just a special case of the first rule as is shown in the right of the table.

$$\frac{\langle \phi_x \rangle : A \Rightarrow \forall x.B \quad \pi_x : \forall x.B \Rightarrow B}{\langle \phi_x \rangle \circ \pi_x : A \Rightarrow B} \qquad \frac{\phi : A \Rightarrow B}{\langle \phi_x \rangle : A \Rightarrow \forall x.B}$$

Table 4.6. $\forall$-adjunction

$\exists$-adjunction means $\psi : B \Rightarrow A$ and $[\psi_x] : \exists x.B \Rightarrow A$ can be bi-directionally constructed. If the latter is assumed, then the former can be derived as is shown in the left of the table 4.7. By the equation, the term, $\kappa_x \circ [\psi_x]$, is equivalent to ψ. The opposite direction is just a special case of the first rule as is shown in the right of the table.

4.6.2 Internalising transitions to linear implication

As a step to encoding quantified objects, we alter our encoding of situation theoretic objects by internalising transitions from $\Rightarrow$ to linear implication, $\multimap$. That is, objects encoded in the form of $A \Rightarrow B$ are now encoded as $\mathbf{1} \Rightarrow A \multimap B$ or $\Rightarrow A \multimap B$ by

$$\frac{\kappa_x : B \Rightarrow \exists x.\,B \quad [\psi_x] : \exists x.\,B \Rightarrow A}{\kappa_x \circ [\psi_x] : B \Rightarrow A} \qquad \frac{\psi : B \Rightarrow A}{[\psi_x] : \exists x.\,B \Rightarrow A}$$

Table 4.7. $\exists$-adjunction

suppressing **1**. Such a change does not affect on the use of other connectives although underlying computation becomes complicated by making it closed category.

We can prove the following relation holds: $A \Rightarrow B$ iff $\mathbf{1} \Rightarrow A \multimap B$. Remember the axiom and rule for linear implication (Figure 4.8). The $\Longrightarrow$ direction holds by taking A in the left rule of the table as **1**, B and C as A and B, respectively. The $\Longleftarrow$ direction holds by the following derivation

$$\cfrac{\cfrac{A \Rightarrow \mathbf{1} \otimes A \quad \cfrac{\mathbf{1} \Rightarrow A \multimap B \quad A \Rightarrow A}{\mathbf{1} \otimes A \Rightarrow (A \multimap B) \otimes A}}{A \Rightarrow (A \multimap B) \otimes A} \qquad (A \multimap B) \otimes A \Rightarrow B}{A \Rightarrow B}$$

Adjointness of $\multimap$ and $\otimes$:

$$\frac{\phi : A \otimes B \Rightarrow C}{cur_{A,B,C(\phi)} : A \Rightarrow B \multimap C} \qquad ev_{A,B} : (A \multimap B) \otimes A \Rightarrow B$$

Table 4.8. The axiom and rule for linear implication, $\multimap$.

As a result of the modification, derivations becomes long. For example, emulating *cut* takes longer steps as demonstrated below, but possible. Hereinafter, we may suppress **1** in the antecedent for simplicity.

$$\cfrac{\cfrac{\cfrac{\mathbf{1} \Rightarrow A \multimap B}{A \Rightarrow B}\text{ as above} \quad \cfrac{\mathbf{1} \Rightarrow B \multimap C}{B \Rightarrow C}\text{ as above}}{A \Rightarrow C}}{\mathbf{1} \Rightarrow A \multimap C}$$

4.6.3 The construction of quantified formulas

The advantage of the quantification presented is that it makes it possible to quantify semantic objects over assignments because propositions representing assignments can be parameterised. If we regard formulas encoding assignments as witnesses, we can regard the construction of universally quantified formulas as the procedure to gather witnesses supporting the formula.

In this section, we employ the encoding we have proposed for naive situation theory. An infon such as $\langle\!\langle$woman, a; 1$\rangle\!\rangle$ is then encoded into the π-calculus as processes $P =_{\text{def}}$

$\overline{woman}\langle r\rangle \mid \overline{r}\langle \mathrm{w}_1\rangle$. To express the meaning of a sentence, *"A woman sings a song"*, we need to discuss how it should be encoded, however. Traditionally, the sentence has been translated in classical logic to (4.73) whilst the sentence, *"Every woman sings a song"*, is translated to either (4.74) or (4.75), depending on interpretation.

$$\exists x \exists y.\,[woman(y) \land song(x) \land sing(y,x)] \tag{4.73}$$

$$\forall y.\,[woman(y) \rightarrow \exists x.\,[song(x) \land sing(y,x)]] \tag{4.74}$$

$$\exists x.\,[song(x) \land \forall y.\,[woman(y) \rightarrow sing(y,x)]] \tag{4.75}$$

The difference between (4.73) and (4.74) has often been criticised as artificial because one cannot recognise such a syntactic difference in original sentences. The reason why one has to use implication, $\rightarrow$, comes from the semantics of classic logic. The formula, $P \rightarrow Q$, is equivalent to $\neg P \lor Q$ thus to $\neg(P \land \neg Q)$. One may say that what the formula does is after all to check that no cases satisfy $P \land \neg Q$ and that it does not care about at all if any women certainly exist. On the other hand, the linear implication, $\multimap$, is stronger than the classical implication in that it demands there is a term or proof supporting the formula, $P \multimap Q$. The characteristic point of linear implication leads to a different interpretation of sentences including *"Every"* and *"A"*.

Imagine how the case where a woman, w_1, sings a song, s_1, can be represented as a process. We can without doubt encode the assignments to w_1 and s_1 as a process such as $\overline{r}\langle w_1\rangle$ and $\overline{u}\langle s_1\rangle$, respectively. But there are two ways to encode the information about woman, sing, and song. One way is to encode it as (4.76), analogous to (4.73). The process may randomly either emit r, the reference to the woman w_1, through *woman* channel, emit r and u, the reference to s_1, through *sing* channel, or emit u through *song* channel. The other way (4.77) is, focussing on *woman*, to emit r through *woman* channel, then emit other references, randomly.

$$(\overline{woman}\langle r\rangle + \overline{sing}\langle r,u\rangle + \overline{song}\langle u\rangle) \mid \overline{r}\langle w_1\rangle \mid \overline{u}\langle s_1\rangle \tag{4.76}$$

$$\overline{woman}\langle r\rangle.(\overline{sing}\langle r,u\rangle + \overline{song}\langle u\rangle) \mid \overline{r}\langle w_1\rangle \mid \overline{u}\langle s_1\rangle \tag{4.77}$$

When translated into the logic, they may be represented as (4.78) and (4.79), respectively. In the translation, we omit processes encoding assignments for simplicity.

$$\phi : \mathbf{1} \Rightarrow \overline{woman}\langle r\rangle \& \overline{sing}\langle r, u\rangle \& \overline{song}\langle u\rangle \tag{4.78}$$

$$\psi : \overline{woman}\langle r\rangle \Rightarrow \overline{sing}\langle r, u\rangle \& \overline{song}\langle u\rangle \tag{4.79}$$

Of these interpretations, we prefer (4.79) to (4.78) because we would like to regard the meaning of quantified sentences as *constraint*. A sentence such as *"A woman sings a song"* is for example understood as stating a constraint between a situation where a woman exists and another where she sings a song. Such a constraint is true if there is at least one connection, ψ, supporting the constraint, but not all connections need to support it. Universally quantified sentences such as *"Every woman sings a song"* can, on the other hand, be regarded as stating stronger constraints, that is, any situation where a woman exists leads necessarily to another situation where she sings a song.

To see quantifier relation as constraints seems to be natural, but we need to elaborate it further to express the difference between the meanings (4.74) and (4.75). We start with the explanation about what object can be quantified over. The above expression, (4.79), is expressed as (4.80), when internalised to $\multimap$. We add to it the formulas representing assignments.

$$\psi : \Rightarrow (\overline{woman}\langle r\rangle \multimap (\overline{sing}\langle r, u\rangle \& \overline{song}\langle u\rangle)) \otimes \overline{r}\langle w_1\rangle \otimes \overline{u}\langle s_1\rangle \tag{4.80}$$

To construct the existentially quantified expression, we parameterise the formulas encoding assignments, $\overline{r}\langle w_1\rangle$ and $\overline{u}\langle s_1\rangle$, to y and x, respectively, and quantify it over them. Since we apply the $\exists$-introduction rule twice to it, we obtain the term shown in (4.81), which may be thought of as a witness supporting the claim that there is a woman who sings a song by recording the replacement of y and x.

$$(\psi \circ \kappa_y^{\overline{r}\langle w_1\rangle}) \circ \kappa_x^{\overline{u}\langle s_1\rangle} : \Rightarrow \exists x \exists y. [(\overline{woman}\langle r\rangle \multimap (\overline{sing}\langle r, u\rangle \& \overline{song}\langle u\rangle)) \otimes y \otimes x] \tag{4.81}$$

From traditional model-theoretic point of view, the truth condition of the sentence is captured in terms assignments, that is, whether there is an assignment to the woman and song satisfying the constraint that the woman sings the song.

In the example, the order of replacement of y and x does not matter, but it matters when we construct universally quantified expressions. As we have seen at the beginning

of this subsection, two interpretations are possible for the sentence, *"Every woman sings a song"*. One case is such that some women sing different songs, and another is such that they sing the same song. We start with the first case. To encode the case as processes, we express them as follows, where each woman, w_i, sings a particular song, s_i.

$$\begin{array}{l}
(\overline{woman}\langle r\rangle.(\overline{sing}\langle r, u\rangle + \overline{song}\langle u\rangle) \mid \overline{r}\langle w_1\rangle \mid \overline{u}\langle s_1\rangle) \quad + \\
(\overline{woman}\langle r\rangle.(\overline{sing}\langle r, u\rangle + \overline{song}\langle u\rangle) \mid \overline{r}\langle w_2\rangle \mid \overline{u}\langle s_2\rangle) \quad + \\
\qquad\qquad \vdots \\
(\overline{woman}\langle r\rangle.(\overline{sing}\langle r, u\rangle + \overline{song}\langle u\rangle) \mid \overline{r}\langle w_n\rangle \mid \overline{u}\langle s_n\rangle)
\end{array}$$

We can construct the formula, $\forall y \exists x. (\overline{woman}\langle r\rangle \multimap \overline{sing}\langle r, u\rangle \& \overline{song}\langle u\rangle) \otimes y \otimes x$, corresponding to $\forall y \exists x. woman(y) \rightarrow sing(y, x) \wedge song(x)$, as is shown in the upper part of the table 4.9. To transform the formulas corresponding to each case in the same shape except to the assignment of woman, $\overline{r}\langle w_i\rangle$, we transform them into an existentially quantified formula. Then, we construct the universally quantified formula by conjoining them with respect to the assignment to woman.

The other case where every woman sings the same song can be expressed as processes such as

$$\begin{array}{l}
(\overline{woman}\langle r\rangle.(\overline{sing}\langle r, u\rangle + \overline{song}\langle u\rangle) \mid \overline{r}\langle w_1\rangle \mid \overline{u}\langle s\rangle) \quad + \\
(\overline{woman}\langle r\rangle.(\overline{sing}\langle r, u\rangle + \overline{song}\langle u\rangle) \mid \overline{r}\langle w_2\rangle \mid \overline{u}\langle s\rangle) \quad + \\
\qquad\qquad \vdots \\
(\overline{woman}\langle r\rangle.(\overline{sing}\langle r, u\rangle + \overline{song}\langle u\rangle) \mid \overline{r}\langle w_n\rangle \mid \overline{u}\langle s\rangle)
\end{array}$$

To construct the formula, $\Rightarrow \exists x \forall y. (\overline{woman}\langle r\rangle \multimap \overline{sing}\langle r, u\rangle \& \overline{song}\langle u\rangle) \otimes y \otimes x$, corresponding to the other interpretation, $\exists x \forall y. woman(y) \rightarrow sing(y, x) \wedge song(x)$, we first quantify all formulas universally over the assignment to woman, then quantify it existentially over the assignment to song, $\overline{u}\langle s\rangle$. Note such an operation is impossible in the first case because the formulas encoding assignments to song, $\overline{u}\langle s_i\rangle$, are different from each other.

4.7 Discussion

4.7.1 Reference as first-class object

When we apply the ideas from the π-calculus to natural language semantics, one of the most beneficial point is in *mobility* because it enables one to treat roles as first-class object. The approach can for example simplify Cooper's analysis of Frege's Hesperus & Phosphorus puzzle using EKN [Coo93a]. We briefly review how the puzzle was analysed:

The construction for the reading of $\forall y\, \exists x.\, woman(y) \rightarrow sing(y,x) \wedge song(x)$:

For any formula, ϕ_i, it holds that

$$\frac{\phi_1 :\Rightarrow (\overline{woman}\langle r\rangle \multimap \overline{sing}\langle r,u\rangle \& \overline{song}\langle u\rangle) \otimes \overline{r}\langle w_1\rangle \otimes \overline{u}\langle s_1\rangle \quad \kappa_x^{\overline{u}\langle s_1\rangle} : (\overline{woman}\langle r\rangle \multimap \overline{sing}\langle r,u\rangle \& \overline{song}\langle u\rangle) \otimes \overline{r}\langle w_1\rangle \otimes \overline{u}\langle s_1\rangle \Rightarrow \exists x.\, (\overline{woman}\langle r\rangle \multimap \overline{sing}\langle r,u\rangle \& \overline{song}\langle u\rangle) \otimes \overline{r}\langle w_1\rangle \otimes x}{\phi_1 \circ \kappa_x^{\overline{u}\langle s_1\rangle} :\Rightarrow \exists x.\, (\overline{woman}\langle r\rangle \multimap \overline{sing}\langle r,u\rangle \& \overline{song}\langle u\rangle) \otimes \overline{r}\langle w_1\rangle \otimes x}$$

Therefore, it follows that

$$\frac{\phi_1 \circ \kappa_x^{\overline{u}\langle s_1\rangle} :\Rightarrow \exists x.\, (\overline{woman}\langle r\rangle \multimap \overline{sing}\langle r,u\rangle \& \overline{song}\langle u\rangle) \otimes \overline{r}\langle w_1\rangle \otimes x \;\cdots\cdots\; \phi_n \circ \kappa_x^{\overline{u}\langle s_n\rangle} :\Rightarrow \exists x.\, (\overline{woman}\langle r\rangle \multimap \overline{sing}\langle r,u\rangle \& \overline{song}\langle u\rangle) \otimes \overline{r}\langle w_n\rangle \otimes x}{\langle (\phi_1 \circ \kappa_x^{\overline{u}\langle s_1\rangle})_y^{\overline{r}\langle w_1\rangle} \cdots (\phi_n \circ \kappa_x^{\overline{u}\langle s_n\rangle})_y^{\overline{r}\langle w_n\rangle} \rangle : \forall y\, \exists x.\, (\overline{woman}\langle r\rangle \multimap \overline{sing}\langle r,u\rangle \& \overline{song}\langle u\rangle) \otimes y \otimes x}$$

The construction for the reading of $\exists x\, \forall y.\, woman(y) \rightarrow sing(y,x) \wedge song(x)$:

It holds that

$$\frac{\psi_1 :\Rightarrow (\overline{woman}\langle r\rangle \multimap \overline{sing}\langle r,u\rangle \& \overline{song}\langle u\rangle) \otimes \overline{r}\langle w_1\rangle \otimes \overline{u}\langle s\rangle \;\cdots\cdots\; \psi_n :\Rightarrow (\overline{woman}\langle r\rangle \multimap \overline{sing}\langle r,u\rangle \& \overline{song}\langle u\rangle) \otimes \overline{r}\langle w_n\rangle \otimes \overline{u}\langle s\rangle}{\langle {\psi_1}_y^{\overline{r}\langle w_1\rangle} \cdots {\psi_n}_y^{\overline{r}\langle w_1\rangle} \rangle :\Rightarrow \forall y.\, (\overline{woman}\langle r\rangle \multimap \overline{sing}\langle r,u\rangle \& \overline{song}\langle u\rangle) \otimes y \otimes \overline{u}\langle s\rangle}$$

Since

$$\kappa_x^{\overline{u}\langle s\rangle} : \forall y.\, (\overline{woman}\langle r\rangle \multimap \overline{sing}\langle r,u\rangle \& \overline{song}\langle u\rangle) \otimes y \otimes \overline{u}\langle s\rangle \Rightarrow \exists x\, \forall y.\, (\overline{woman}\langle r\rangle \multimap \overline{sing}\langle r,u\rangle \& \overline{song}\langle u\rangle) \otimes y \otimes x$$

It follows that

$$\langle {\psi_1}_y^{\overline{r}\langle w_1\rangle} \cdots {\psi_n}_y^{\overline{r}\langle w_1\rangle} \rangle \circ \kappa_x^{\overline{u}\langle s\rangle} :\Rightarrow \exists x\, \forall y.\, (\overline{woman}\langle r\rangle \multimap \overline{sing}\langle r,u\rangle \& \overline{song}\langle u\rangle) \otimes y \otimes x$$

Table 4.9. The construction of quantified formulas

> Frege [Fre93] pointed out that the ancients did not realize that Hesperus (the evening star) was the same heavenly body as Phosphorus (the morning star) and that this created a problem for semantics. The ancients presumably did realize that Hesperus was identical with itself. But if Phosphorus is the same thing as Hesperus, how come they did not realize that too? The answer given here is a rather obvious and simple one. What the ancients did not know had to do with the relationship between linguistic labels and objects in the world. They did not know that they were linking the two words 'Hesperus' and 'Phosphorus' to the same object, thought in fact that is what they were doing.[Coo93a]

The meaning of the sentence, *"Hesperus is not Phosphorus"*, uttered by the ancients then can be expressed in EKN as a relation such as[9]

$$\begin{array}{|l|}\hline Hesperus \rightarrow X, \\ Phosphorus \rightarrow Y \\ \hline \\ \text{is-not}(X, Y) \\ \\ \hline \end{array} \tag{4.82}$$

which means that Hesperus is not identical with Phosphorus. The representation should be contrasted with other meanings such as

(4.83) a. Hesperus is not identical with Hesperus
b. Hesperus is not identical with itself

which may be expressed in EKN, respectively, as

$$\begin{array}{|l|}\hline Hesperus_1 \rightarrow X, \\ Hesperus_2 \rightarrow Y \\ \hline \\ \text{is-not}(X, Y) \\ \\ \hline \end{array} \tag{4.84}$$

$$\begin{array}{|l|}\hline Hesperus \rightarrow X \\ \hline \\ \text{is-not}(X, X) \\ \\ \hline \end{array} \tag{4.85}$$

When we encode these three objects into the π-calculus, the first one (4.82) may be encoded as (4.86), the second one (4.84) as (4.87), and third one (4.85) as (4.88), by

[9] It should be expressed more precisely as a situation-theoretic type, but we treat it for simplicity as if it were a relation.

directly encoding linguistic labels, 'Hesperus' and 'Phosphorus' as channels, h and p, respectively:

$$\overline{is\text{-}not}\langle h, p\rangle \tag{4.86}$$
$$\overline{is\text{-}not}\langle h_1, h_2\rangle \tag{4.87}$$
$$\overline{is\text{-}not}\langle h, h\rangle \tag{4.88}$$

We have to be cautious in distinguishing (4.84) from (4.85), which leads to distinguishing the two propositions, (4.83a) and (4.83b), because (4.84) assumes that the same proper name, *Hesperus*, can be used as different linguistic labels, $Hesperus_1$ and $Hesperus_2$. If we did not admit that the same proper name to be used as different roles, the object (4.84) would have been written as (4.89). Then, there is no ground to distinguish (4.84) and (4.85), and both could be encoded as (4.88).

$$\begin{array}{|l|}\hline \quad Hesperus \to X, \\ Hesperus \to Y \\ \hline \\ \text{is-not}(X, Y) \\ \\ \hline \end{array} \tag{4.89}$$

The representation (4.84) may be acceptable if we extend the idea that *"linguistic expressions as labels"* to *linguistic actions as labels.* One can think of the case where the two expressions of 'Hesperus' can be uttered with different meanings. Suppose a person who is ignorant of astronomy mistakenly believed a wrong star was Hesperus for a long time, say from his childhood. One day, he may suddenly notice that he was wrong when one of his friends happens to point out the star, saying *"The star is Hesperus."* Then, the person might say, *"Ah, Hesperus is not Hesperus!"*. The meanings of the first and second occurences of 'Hesperus' in this case are different. If we pursue the idea further, roles can be more complex objects. The move has been initiated by Cooper [Coo93a]. In such an approach, the example can be analysed as (4.90), where X corresponds to the referent of the first utterance of 'Hesperus' and Y corresponds to that of the second utterance.

$$\begin{array}{|l|}\hline \quad \langle\, ref, Hesperus_1 \,\rangle \to X, \\ \langle\, ref, Hesperus_2 \,\rangle \to Y \\ \hline \\ \text{is-not}(X, Y) \\ \\ \hline \end{array} \tag{4.90}$$

How such an extention can be dealt with in our approach? Mobility can play an important role then. To implement and refine the idea that linguistic actions as labels, we regard such actions as a sort of process. We express the referring processes of the first and second utterances as independent input processes, $ref_1(h_1)$ and $ref_2(h_2)$, expressing them in molecule form as (4.91).

$$[\, ref_1(h_1), ref_2(h_2), \overline{is\text{-}not}\langle h_1, h_2\rangle \,] \tag{4.91}$$

These two input processes will import some indices through ref_1 and ref_2 and replace them for h_1 and h_2, respectively. Such input processes can be more complex. The channel, ref_1, itself might be imported through another channel. But our approach basically provides for the necessary machinery to implement more complex referring processes. Notice it is mobility that makes it possible to abstract objects over roles and import them. Since roles or channels are treated as first-class object, we can import and export them freely from or to other objects. Our approach can therefore give a clear insight on what is going on when we introduce these complex indices into EKN.

4.7.2 Mental states as processes

Our encoding of situation-theoretic objects into the π-calculus as processes may shed a new light on situation-theoretic analysis of mental states. Mental states are thought of as a sort of situation which can support infons. The person's mental state, who realised he was wrong in identifying Hesperus, can for example be expressed as (4.92). The first argument of the relation *believe*, a, is the agent who is in the mental state, the second is the internal state of a, the third is his connections to the world, and the fourth indicates time. ms_a is his mental state conceived of as a situation. If we understand his utterance, *"Ah, Hesperus is not Hesperus!"*, as *"(The star he has been regarding as) Hesperus is not (actually) Hesperus"*, the first occurrence of 'Hesperus' denotes a wrong heavenly body, m, while the second the correct one, v (Venus).

$$\boxed{\begin{array}{l} ms_a \\ \text{believe}(a, \begin{array}{|l|}\hline \langle\, ref, Hesperus_1 \,\rangle \to X, \\ \langle\, ref, Hesperus_2 \,\rangle \to Y \\ \hline \text{is-not}(X,Y) \\ \hline \end{array} , \begin{bmatrix} \langle\, ref, Hesperus_1 \,\rangle \to m \\ \langle\, ref, Hesperus_2 \,\rangle \to v \end{bmatrix}, t\ ) \end{array}} \tag{4.92}$$

To encode such a mental state as processes, we assume the heavenly bodies, m and v, are accessible through channels r_1 and r_2. They are thus encoded as output processes such as $\overline{r_1}\langle m\rangle$ and $\overline{r_2}\langle v\rangle$, respectively. We disregard the relation, *believe*, and the agent a for the moment. His connections to these heavenly bodies are established first by emitting these channels through ref_1 and ref_2, then by replacing them for h_1 and h_2. After then, the agent can get access to m and v through received channels. The following shows the transition. The first molecule encodes his initial internal state, and the second his connections.

$$\begin{array}{ll} & [\,[\,ref_1(h_1), ref_2(h_2), \overline{\textit{is-not}}\langle h_1, h_2\rangle\,], (\nu\, r_1, r_2)[\,\overline{ref_1}\langle r_1\rangle, \overline{ref_2}\langle r_2\rangle, \overline{r_1}\langle m\rangle, \overline{r_2}\langle v\rangle\,]\,] \\ {}_\tau\!\searrow & (\nu\, r_1, r_2)[\,ref_2(h_2), \overline{\textit{is-not}}\langle r_1, h_2\rangle, \overline{ref_2}\langle r_2\rangle, \overline{r_1}\langle m\rangle, \overline{r_2}\langle v\rangle\,]*\{{}^{r_1}/_{h_1}\} \\ {}_\tau\!\searrow & (\nu\, r_1, r_2)[\,\overline{\textit{is-not}}\langle r_1, r_2\rangle, \overline{r_1}\langle m\rangle, \overline{r_2}\langle v\rangle\,]*\{{}^{r_1}/_{h_1}, {}^{r_2}/_{h_2}\} \end{array}$$

The actual referring and anchoring processes must be more complex than just demonstrated above, but it suffices to show that the idea to regard mental states as processes fits very well with our intuition about mental states. They can be better captured as processes rather than situations. Our approach can refine the situation theoretic analysis of mental states in natural way.

The remaining question at hand is how to encode the infon including another complex object as its argument. To start with the investigation, we argue that *relations*, too, can be indexed as object. Observe we often need to specify the meaning of relations or properties in conversation. For example, in the excerpt shown in the figure 4.6, the route follower asks the giver to clarify the meaning of `chotto` (a little bit). Based on the observation, we think relations can be parameterised with roles.

If we allow to abstract semantic objects over parameters replacing relations, the encoding of infon, $\langle\!\langle$smile, j; 1$\rangle\!\rangle$, is modified from (4.93) to (4.94), where r_0 is the channel to the relation, smile. There appears to be a new channel, i, however, for which we need some explanation. We would like to regard it as a channel to get access to the infon. Once we admit such channels, we are ready to encode complex objects.

G$_1$:	`de-e, hanshu-kurai sita-ra`	`chotto hidari-ni iku-n-desu-yo`
	and, after you go around about half	*you go leftward a little bit*
F$_1$:	`chotto-tte dono-kurai?`	
	What do you mean by "a little bit"?	

Figure 4.6. An excerpt from the Japanese Map Task Corpus (4)

$$(\nu\, r_1)[\,\overline{smile}\langle r_1\rangle, \overline{r_1}\langle \text{j}\rangle\,] \tag{4.93}$$

$$(\nu\, i, r_0, r_1)[\,\overline{i}\langle r_0, r_1\rangle, \overline{r_0}\langle \text{smile}\rangle, \overline{r_1}\langle \text{j}\rangle\,] \tag{4.94}$$

We encode as an example the following infon (4.95) as processes, the infon that Mary believes John smiles.

$$\boxed{\begin{array}{l} ms_{Mary} \\ \text{believe}(m, \begin{array}{|l|}\hline r_0 \to R,\ \ r_1 \to X \\ \hline R(X) \\ \hline \end{array} , \begin{bmatrix} r_0 \to \text{smile} \\ r_1 \to \text{j} \end{bmatrix}, \text{t}) \end{array}} \tag{4.95}$$

Assuming the information about John's smiling is available through a channel, i, the infon can be encoded as

$$(\nu\, i, j, r_2, r_3)[\,\overline{j}\langle r_2, r_3, i\rangle, \overline{r_2}\langle believe\rangle, \overline{r_3}\langle \text{m}\rangle\,] \tag{4.96}$$

where j is the channel to get access to the infon, r_2 is the index to the relation, believe, and r_3 the index to Mary, m. Since the information about John's smiling can be encoded as (4.94), the molecule we obtain as the result is the merged one of them (4.97), which can be depicted in datagraph as is shown in the figure 4.7.

$$(\nu\, i, j, r_0, r_1, r_2, r_3)[\,\overline{j}\langle r_2, r_3, i\rangle, \overline{i}\langle r_0, r_1\rangle, \overline{r_2}\langle believe\rangle, \overline{r_3}\langle \text{m}\rangle, \overline{r_0}\langle \text{smile}\rangle, \overline{r_1}\langle \text{j}\rangle\,] \tag{4.97}$$

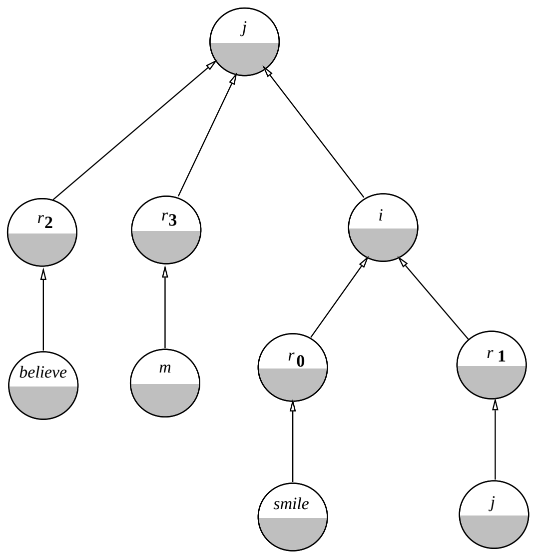


Figure 4.7. The dataflow graph of *"Mary believes that John smiles"*

4.7.3 Shared-situations as processes

The channels introduced to index infons in the above enables one to encode a particular use of situations to describe common knowledge. In [Bar89], Barwise proposed a shared-situation approach to study common knowledge, which developed the idea proposed by Clark and Marshall in [CM92]. In maptask, such knowledge is central because the goal of the task is to establish common knowledge such that

- the giver knows the follower reaches the goal,
- the follower knows the giver knows he(f.) reaches the goal,
- the giver knows the follower knows he(g.) knows he(f.) reaches the goal,
- the follower knows the giver knows he(f.) knows he(g.) knows he(f.) reaches the goal, and so on.

In fact, the participants have to establish their common knowledge as to every item in the map having to do with the route.

The shared-situation approach can be summarised as follows:

- $s \models \sigma$
- $s \models g$ knows s
- $s \models f$ knows s

In the definition, σ means the fact established as common knowledge, s is the shared situation, g and f are the giver and follower, respectively. The point of the definition is that both of them have the access to the situation s such that it supports the fact and their states of communication, where s itself is involved in the infon.

In that approach, one can see the situation s plays two roles, one of which is to support infons, the other of which is to provide for access to itself in order to extract information.[10] Of these, the latter role can be implemented in our approach by regarding the channels to infons as the access to situations. Suppose the infon, σ, is accessible through a channel, s. Then, the definition can be encoded as processes such as

$$\begin{aligned}
&(\nu\, s)(\overline{s}\langle\sigma\rangle\ + \\
&(\nu\, r_1, r_2, r_3)(\overline{s}\langle r_1, r_2, r_3\rangle\ \mid \overline{r_1}\langle knows\rangle\ \mid \overline{r_2}\langle g\rangle\ \mid \overline{r_3}\langle s\rangle) + \\
&(\nu\, r_4, r_5, r_6)(\overline{s}\langle r_4, r_5, r_6\rangle\ \mid \overline{r_4}\langle knows\rangle\ \mid \overline{r_5}\langle f\rangle\ \mid \overline{r_6}\langle s\rangle))
\end{aligned}$$

where σ is a sequence of channels to a relation and arguments. The encoding may not give much insight, but one can see the structure of the common knowledge when it

[10] Such a view on situations is adopted in designing PROSIT [NPS91].

is depicted in dataflow graph as is shown in the figure 4.8. In the figure, the arcs r_i connected to the boxes mean they are sorted out in that order. Now, one can observe two cyclic paths in the graph. One is the path, $r_3 \leftarrow s \leftarrow r_3 \leftarrow \cdots$, and the other is the path, $r_6 \leftarrow s \leftarrow r_6 \leftarrow \cdots$. Such cyclic paths enable other processes to extract those items of information as if they are infinitely nested when they are replicable. That is, one can follow a path such as $r_3 \leftarrow s \leftarrow r_6 \leftarrow s \leftarrow \cdots$, which can be thought to represent common knowledge between the giver and follower.

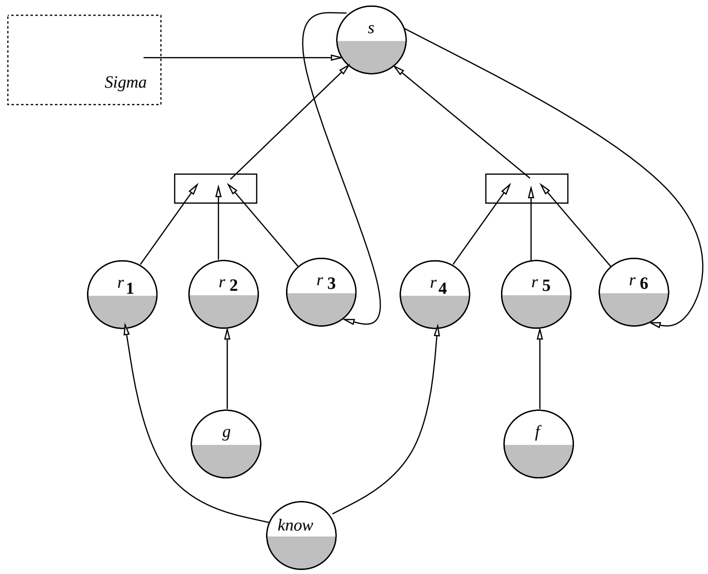


Figure 4.8. The dataflow graph of a shared situation

The process-algebraic view can also shed a new light on the relation between different structures of common knowledge. Let us see another example of common knowledge, which is more complex than the previous one.

- $s_1 \models \sigma$
- $s_1 \models g$ knows s_1
- $s_1 \models f$ knows s_2
- $s_2 \models \sigma$

- $s_2 \models g$ knows s_1
- $s_2 \models f$ knows s_2

There should be models satisfying this definition, but the previous one. How can such a difference be captured? We first depict the definition in dataflow graph (Figure 4.9). One can see the structure of the common knowledge is fairly enriched. The cyclic paths are now extended to the following three, where the third path did not exist in the previous definition, the cycle emphasised with bold line in the figure. It is the path that establishes common knowledge.

- $r_3 \leftarrow s_1 \leftarrow r_3 \leftarrow s_1 \leftarrow \cdots$,
- $r_6 \leftarrow s_2 \leftarrow r_6 \leftarrow s_2 \leftarrow \cdots$
- $r_6 \leftarrow s_2 \leftarrow r_3 \leftarrow s_1 \leftarrow r_6 \leftarrow s_2 \leftarrow \cdots$

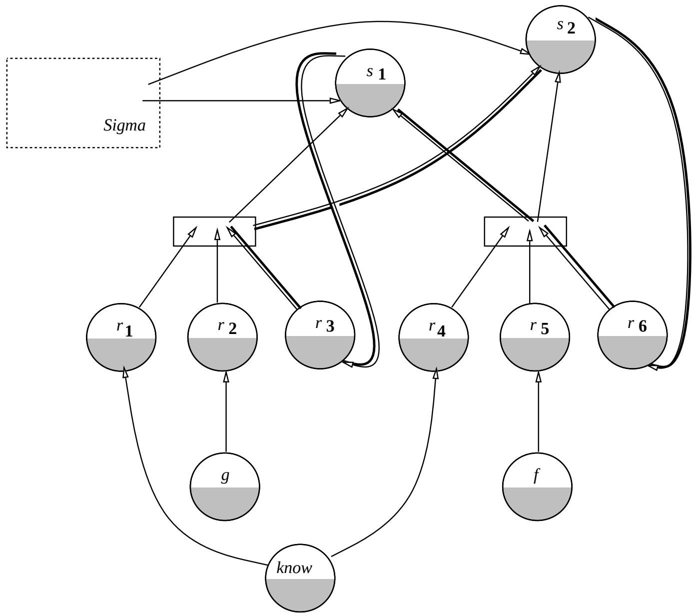


Figure 4.9. The dataflow graph of a more complex shared situation

The structures are different, but what relation would hold between them? They are different, but still have much in common. To capture the relation, we employ the notion of *simulation*. Observe the structure (Figure 4.9) becomes equivalent to that shown in Figure 4.8 when the two nodes, s_1 and s_2, are merged and renamed to s. Given this observation, we say the system, P (Figure 4.9), *simulates* another, Q (Figure 4.8), under the operation and express the relation as $Q < P$.[11]

4.8 Conclusion

In the chapter, we have applied the ideas from the π-calculus to studying Situation Semantics. Our aim was to base Situation Semantics to a computational ground of communication and concurrency and to linear logic. In constructing semantic objects for primitive Situation Semantics, communication and concurrency play an important role to implement simultaneous abstractions and applications. The π-calculus further enables us to encode situation-theoretic objects more effectively thanks to mobility. As references are treated as first-class object, we can encode abstract objects as objects without assignments. Mobility also enables us to encode more complex abstractions and applications involving referring actions as processes. Quantification relation has been defined constructively, where terms serve as witnesses. We finally discussed the issues of mental states and common knowledge.

Through the chapter, we have also presented a way to bridge the gap between the research on information flow, i.e., Channel Theory, and the project to build a well-formed semantic universe for linguistic applications, i.e., Situation Theoretic Discourse Representation Theory [Coo93a, Coo93b], which have been so far studied rather independently. We have encoded various situation-theoretic objects into the π-calculus and have shown in the previous chapter how the calculus can be related with a combinatorial intuitionistic linear logic, which can be regarded as stating the principle of information flow.

Let us review the principle of information flow proposed by Barwise [Bar93] shown in the table 4.10. We can relate these postulates with the axioms and rules of the logic as follows:

- 'Xerox Principle' corresponds to sequential composition, $\circ$ (cut).
- 'Logic as Information Flow' states our view itself. The slight difference is that we write $s : t_1 \Rightarrow t_2$ for $s : t_1 \longrightarrow s : t_2$.

[11] Bisimulation relation $A \sim B$ can be defined as the relation such as $A < B$ and $A > B$. The relation is strongly bisimular when we strictly map a graph to another. If we disconsider parts corresponding to τ, then the relation is weekly bisimular. We do not go into detail of these notions as it is outside our scope.

- 'Addition of Information' corresponds to parallel composition, $\otimes$ (multiplicative conjunction).
- 'Exhaustive Cases' can be derived using the rules for coproducts and cut.

$$\frac{\phi : t_1 \Rightarrow t_2 \oplus t_2' \qquad \dfrac{\psi_1 : t_2 \Rightarrow t_3 \quad \psi_2 : t_2' \Rightarrow t_3}{[\psi_1, \psi_2] : t_2 \oplus t_2' \Rightarrow t_3}}{\phi \circ [\psi_1, \psi_2] : t_1 \Rightarrow t_3}$$

- 'Contraposition' does not hold in our system because the logic is intuitionistic. It will, however, hold if we make the logic classical by introducing $⅋$, the dual of $\otimes$ with its neutral element, $\perp$.

1. Xerox Principle:
$$\frac{s_1 : t_1 \longrightarrow s_2 : t_2 \quad s_2 : t_2 \longrightarrow s_3 : t_3}{s_1 : t_1 \longrightarrow s_3 : t_3}$$
2. Logic as Information Flow:
$$\frac{t_1 \vdash t_2}{s : t_1 \longrightarrow s : t_2}$$
3. Addition of Information:
$$\frac{s_1 : t_1 \longrightarrow s_2 : t_2 \quad s_1 : t_1' \longrightarrow s_2 : t_2'}{s_1 : t_1 \wedge t_1' \longrightarrow s_2 : t_2 \wedge t_2'}$$
4. Exhaustive Cases:
$$\frac{s_1 : t_1 \longrightarrow s_2 : t_2 \vee t_2' \quad s_2 : t_2 \longrightarrow s_3 : t_3 \quad s_2 : t_2' \rightarrow s_3 : t_3}{s_1 : t_1 \longrightarrow s_3 : t_3}$$
5. Contraposition:
$$\frac{s_1 : t_1 \longrightarrow s_2 : t_2}{s_2 : \neg t_2 \longrightarrow s_1 : \neg t_1}$$

Table 4.10. The principle of information flow: Barwise's postulates

Closer look will, however, reveal subtle differences. Table 4.11 shows the axioms for Channel Algebra [MS94] except those for the invertible. In the table, $c_1 \circ c_2$ means sequential composition, $c_1 \wedge c_2$ parallel composition, and 1 identity. The axioms, (i) and (ii), hold in our system, too, but not (iii) because the logic is *linear*. The axioms, (iv) and (v), hold in our system, too, by category axioms though we use *id* rather than 1. The axioms, (vi) and (vii), do not hold, however. Instead, we have a natural isomorphism, $\gamma \circ (\phi \otimes \psi) = (\psi \otimes \phi) \circ \gamma$. The difference comes from the lack of (iii).

To sum up, comparing our foundation with Channel Theory, ours is slightly different in that it is intuitionistic and linear, but otherwise they share much in common. In fact, Barwise proposed a way to view linear logic as a theory of information flow [Bar92]. We think therefore our approach is still in track of Channel Theory. Although there are still many rooms for further development, we believe our approach is a promising way to construct semantic objects on a rigorous foundation for Situation Semantics.

(i)	$c_1 \wedge c_2 = c_2 \wedge c_1$	$\wedge$ is commutative
(ii)	$c_1 \wedge (c_2 \wedge c_3) = (c_1 \wedge c_2) \wedge c_3$	$\wedge$ is associative
(iii)	$c \wedge c = c$	$\wedge$ is idempotent
(iv)	$c_1 \circ (c_2 \circ c_3) = (c_1 \circ c_2) \circ c_3$	$\circ$ is associative
(v)	$1 \circ c = c \circ 1 = c$	1 is the unit of $\circ$
(vi)	$c_1 \circ (c_2 \wedge c_3) \leq (c_1 \circ c_2) \wedge (c_1 \circ c_3)$	left semi-distribution of $\circ$ over $\wedge$
(vii)	$(c_2 \wedge c_3) \circ c_1 \leq (c_2 \circ c_1) \wedge (c_3 \circ c_1)$	right semi-distribution of $\circ$ over $\wedge$

Table 4.11. The axioms for Channel Algebra

Chapter 5

Parsing as reaction

5.1 Introduction

In this chapter, we present a parser that can utilise different sources of information while analysing syntactic information of sentences. By "different sources", we mean *contexts*, which we have elaborated in the first chapter to *discourse*, *plan-goal*, and *circumstances*. We have claimed then based on our observation that a parser must be designed so that it can extract freely items of information from different sources in parallel. To implement the idea in a parser, we propose a parser that analyses syntactic information as the result of a course of *reactions*. Designed as a reactive system, the parser can spontaneously extract information from environments, not only from sentences uttered, and evolves into another state.

While the reactive parser meets our requirement, it will call in a new problem, too, the problem that we cannot design a particular algorithm to control the global behaviour of the system of interacting processes. Since sentences are parsed as the result of a number of interactions between processes comprising the system, all we can directly program is the behaviour of each process upon interactions. It is therefore desirable that we can ensure that a system as a whole should turn into a particular state in a particular context. We would like for example to know if the state turns into s (sentence) after receiving a sequence of words, e.g., *"a man walks."* The problem is the same as the *reachability* problem for petri-nets, and our approach is effective in studying the problem, too.

The idea of parsing as reactions can be sketched as follows:

- Each word uttered evokes a process, and an utterance of a sentence evokes a set of processes.

- Each process is connected with its neighbours through channels and may exchange feature information with them.
- The communication may leads to creating a new process depending on the information available. The created process is connected with its neighbours through its own channels, too.

The figure 5.1 shows an example of parsing the sentence, *"a man walks."* In the figure, each process encoding a lexical item is depicted as a circle, connected through channels to its right and left neighbours. Each process encodes feature information and may create new processes upon interaction. For example, if the process, Q, encoding **MAN**, extracts feature information of another process, P, encoding the determiner, **A**, and the information meets a condition, then Q generates a new process, Q', encoding a noun phrase. The connections between processes are reconfigured upon the evolution. In this case, the process Q' created is connected to V at the left, the process encoding the end of the previous sentence, and R at the right, the process encoding the lexical information of **WALKS**. The process R creates another process R' corresponding to sentence upon the interaction with Q', which is connected with W, at its right position, which is the process encoding the first item of the next sentence.

The idea to parse sentences as the result of message passing is not novel. The *ParseTalk* model proposed by Bröker, Hahn, and Schacht [SHB94, BHS94, BSSH94] has already investigated how sentences can be parsed through communication between agents, based on Hewitt's actor model [Agh86]. There is also a concurrent constraint programming language, Oz, developed by Gert Smolka and his group at DFKI (German Research Center for Artificial Intelligence) [Smo94b], who have proposed a way to encode feature structures as concurrent processes. Yet many things remain to be investigated. Although the *ParseTalk* model has proposed an interesting approach to parsing, the level of modelling is still informal as they define grammatical information at a level of programming language, whose semantics must be formalised further so that we can infer the behaviour of the parser. The possibility of Oz language is also yet to be exploited to parsing sentences.[1] Our model should serve as a basis to analyse and compare various proposals to concurrent natural language processing.

As our aim is to investigate a general framework to modelling concurrent natural language processing, the parser presented here does not commit to any particular grammar theory. The reason why we employ feature structures to represent lexical information is merely because it is most widely adopted by linguists. The chapter is organised as follows: We present how lexical information can be encoded as a process (§5.2), then go on to presenting how sentences may be parsed as a result of interactions (§5.3), which will

[1]A parser is encoded in Oz, which is found as one of demo programs in the package [Smo94b].

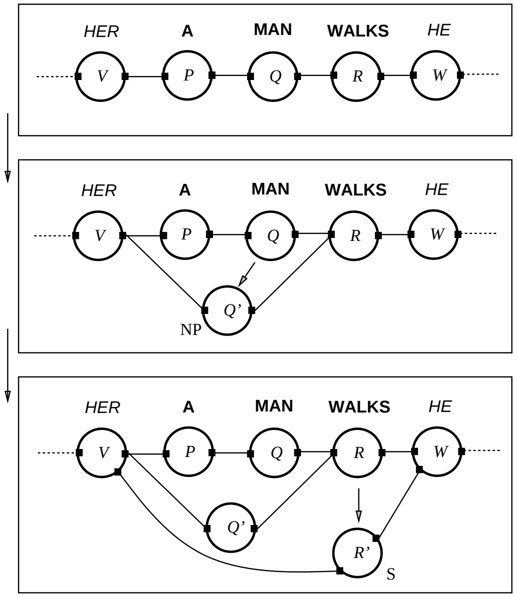


Figure 5.1. Parsing as interactions

be further elaborated in the following section (§5.4). We discuss then logical checking of concurrent parsers, syntax-semantics interface, and structure of feature descriptions based on channel types.

5.2 Feature structures as processes

5.2.1 The basic idea

We start by looking into the simplest form of feature structures. We conceive of a simple feature description (5.1) as (5.2). The additional information, r_0 and r_1, indicate channels through which items of information are available. That is, the whole item of information is accessible through r_0, and *verb*, the value of feature *cat* (category), through r_1.

$$\begin{bmatrix} \mathit{cat:}\ \mathit{verb} \end{bmatrix} \tag{5.1}$$

$$\boxed{r_0}\begin{bmatrix} \mathit{cat:}\ \boxed{r_1}\mathit{verb} \end{bmatrix} \tag{5.2}$$

The feature description (5.2) is encoded into the π-calculus as a process such as

$$(\nu\, r_0, r_1)(!\, c(d).\overline{d}\langle r_0\rangle.r_0(x).[x = \mathtt{cat}]\overline{d}\langle r_1\rangle.\overline{r_1}\langle \mathtt{verb}\rangle)$$

The behaviour of the process can be explained as follows:

- By $c(d).\overline{d}\langle r_0\rangle$, it receives from another process through c a channel replacing d. r_0 is emitted through the channel to provide the process with the access. We assume the channel replacing d is bound name, which ensures that other processes should not receive r_0 through the channel.
- Then, the process receives through r_0 a feature name x to return r_1 through the received channel, if the feature name matches to `cat`.
- Through r_1, it emits `verb`.
- The channels, r_0 and r_1, are private to the process.
- The system will be recreated as many times as requested with the same channels, r_0 and r_1, each time. In practice, this allows several processes to extract information of the process at the same time.

The above encoding shows our basic idea in encoding feature descriptions as processes. One may wonder, as for the way the value is accessed from other processes, why not

the process returns `verb` directly, rather than r_1. That is, why do we not encode it as the following?

$$(\nu\, r_0)(!\, c(d).\overline{d}\langle r_0\rangle.r_0(x).[x = \mathtt{cat}]\overline{d}\langle \mathtt{verb}\rangle)$$

The reason is rather subtle and has to do with typing of channels. We would like to use the received channel d only to send channels, r_0 and r_1. The above encoding violates the principle as d is used to emit as value, `verb`. We would also like to use channels, r_0 and r_1, only for exchanging feature names or values, which we treat as string, not as channels.[2] The type of these channels then can be said to be of exchanging strings. Such a convention contributes to simplifying the type of d, that is, a channel for exchanging channels to exchange strings. We do not go into the discussion on typing of channels [3], but we care about it in our encoding because such a practice will become important when designing models for NLP systems.

5.2.2 Flat structures

We go on to encoding slightly complex feature descriptions. Suppose we represent lexical entry for 'walk' as (5.3). The entry offers at r_0 a new item of information that its phonetic form is *"walk"*.

$$\boxed{r_0}\begin{bmatrix} \mathit{cat:} & \boxed{r_1}\mathit{verb} \\ \mathit{phon:} & \boxed{r_2}\mathit{walks} \end{bmatrix} \tag{5.3}$$

The description can be encoded as below. The first part of receiving a channel to replace d is the same as before, but the process has two choices in evolving into another state, depending on the name replacing x. One of which is the same as before, and the other is to evolve into a process such that it returns r_2 if the received name matches to `phon` and emits `walks` through r_2.

$$\begin{aligned}(\nu\, r_0, r_1, r_2)(\ !\, c(d).\overline{d}\langle r_0\rangle.r_0(x).(\ & [x{=}\mathtt{cat}]\overline{d}\langle r_1\rangle.\overline{r_1}\langle \mathtt{verb}\rangle \\ +\ & [x{=}\mathtt{phon}]\overline{d}\langle r_2\rangle.\overline{r_2}\langle \mathtt{walks}\rangle\ ))\end{aligned}$$

5.2.3 Nested structures

The encoding can be extended to encode nested feature structures. Let us see how the structure (5.4) can be encoded, which represents the agreement information (*agr*) that

[2] That is the reason why we use the different font for them.
[3] See for example [Pie94] to know the type system adopted in PICT.

person (*per*) is third (*3rd*).

$$\boxed{r_0}\begin{bmatrix} \mathit{agr}\!: \boxed{r_3}\begin{bmatrix} \mathit{per}\!: \boxed{r_4}\mathit{3rd} \end{bmatrix} \end{bmatrix} \tag{5.4}$$

The description can be encoded as below. The first part is the same as before, but the channel, r_3, which the process returns through d, does not offer a value but the access to the process for further inquiry. It receives another feature name to replace y at r_3 and returns r_4 if it matches to `per`. The value `3rd` is now offered through r_4.

$$(\nu\, r_0, r_3, r_4)(\,!\,c(d).\overline{d}\langle r_0\rangle.\ r_0(x).[x{=}\mathtt{agr}]\overline{d}\langle r_3\rangle.\ r_3(y).[y{=}\mathtt{per}]\overline{d}\langle r_4\rangle.\ \overline{r_4}\langle \mathtt{3rd}\rangle)$$

By combining these techniques, one can encode a feature description for *"walk"* (5.5) as the process shown below.

$$\boxed{r_0}\begin{bmatrix} \mathit{cat}\!: & \boxed{r_1}\mathit{verb} \\ \mathit{phon}\!: & \boxed{r_2}\mathit{walks} \\ \mathit{agr}\!: & \boxed{r_3}\begin{bmatrix} \mathit{per}\!: & \boxed{r_4}\mathit{3rd} \\ \mathit{num}\!: & \boxed{r_5}\mathit{sing} \end{bmatrix} \end{bmatrix} \tag{5.5}$$

$$\begin{aligned}(\nu\, r_0, r_1, r_2, r_3, r_4, r_5)(\,!\,c(d).\overline{d}\langle r_0\rangle.r_0(x).(&[x{=}\mathtt{cat}]\overline{d}\langle r_1\rangle.\overline{r_1}\langle \mathtt{verb}\rangle \\ &+ [x{=}\mathtt{phon}]\overline{d}\langle r_2\rangle.\overline{r_2}\langle \mathtt{walks}\rangle \\ &+ [x{=}\mathtt{agr}]\overline{d}\langle r_3\rangle.r_3(y).([y{=}\mathtt{per}]\overline{d}\langle r_4\rangle.\overline{r_4}\langle \mathtt{3rd}\rangle \\ &\qquad\qquad + [y{=}\mathtt{num}]\overline{d}\langle r_5\rangle.\overline{r_5}\langle \mathtt{sing}\rangle)))\end{aligned}$$

5.2.4 Shared structures

When describing features, a structure can be shared. Abstractly, one may represent a feature such as (5.6), where the structure $[h : a]$ is shared between f and g.

$$\boxed{r_0}\begin{bmatrix} f\!: \boxed{r_1}\begin{bmatrix} h\!: \boxed{r_2}a \end{bmatrix} \\ g\!: \boxed{r_1} \end{bmatrix} \tag{5.6}$$

As to such a shared structure, a straightforward encoding encounters a problem. If we encode it as below, some other process cannot extract the information $[h : a]$ when it tries to get access to it through g because the information is lost when $[x{=}\mathtt{g}]\overline{d}\langle r_1\rangle$ is chosen to perform.

$$\begin{aligned}(\nu\, r_0, r_1, r_2)(!\,c(d).\overline{d}\langle r_0\rangle.r_0(x).(&[x{=}\mathtt{f}]\overline{d}\langle r_1\rangle.r_1(y).[y{=}\mathtt{h}]\overline{d}\langle r_2\rangle.\overline{r_2}\langle \mathtt{a}\rangle \\ &+ [x{=}\mathtt{g}]\overline{d}\langle r_1\rangle)\end{aligned}$$

One solution is to regard the above structure (5.6) as an abbreviation of (5.7). Then, one can encode it as below, avoiding the problem.

$$\boxed{r_0}\begin{bmatrix} f\!: \boxed{r_1}\begin{bmatrix} h\!: \boxed{r_2}a \end{bmatrix} \\ g\!: \boxed{r_1}\begin{bmatrix} h\!: \boxed{r_2}a \end{bmatrix} \end{bmatrix} \tag{5.7}$$

$$\begin{aligned}(\nu\, r_0, r_1, r_2)(\,!\,c(d).\overline{d}\langle r_0\rangle.r_0(x).(&[x{=}\mathtt{f}]\overline{d}\langle r_1\rangle.r_1(y).[y{=}\mathtt{h}]\overline{d}\langle r_2\rangle.\overline{r_2}\langle \mathtt{a}\rangle \\ &+ [x{=}\mathtt{g}]\overline{d}\langle r_1\rangle.r_1(y).[y{=}\mathtt{h}]\overline{d}\langle r_2\rangle.\overline{r_2}\langle \mathtt{a}\rangle)\end{aligned}$$

The solution does however not capture the meaning of *"shared"* because the latter item of the information is a *copy*. To represent the fact that the same structure is shared, we have to refine our encoding by letting the processes offering the information be independent. When we treat each action as processes independent of others, the feature description is encoded as below. The first part of getting the channel to replace d is the same as before, but then the process evolves into three processes. One of which is to receive a name at r_0 and emits r_1 if the name matches to `f` or `g`. The second process receives a name at r_1 and emits r_2 if it matches to `h`. The third emits `a` through r_2. Since the actions for emitting the channel r_2 and for emitting the value `a` are performed by independent processes, they do not disappear no matter which name is received through r_0.

$$\begin{aligned}(\nu\, r_0, r_1, r_2)(\,!\,c(d).\overline{d}\langle r_0\rangle.(&r_0(x).([x{=}\mathtt{f}]\overline{d}\langle r_1\rangle + [x{=}\mathtt{g}]\overline{d}\langle r_1\rangle) \\ &|\, r_1(y).[y{=}\mathtt{h}]\overline{d}\langle r_2\rangle \\ &|\, \overline{r_2}\langle \mathtt{a}\rangle))\end{aligned}$$

Pursuing the approach further, one may think of separating the process, $c(d).\overline{d}\langle r_0\rangle$, too, as below. But this encoding does not work because d in $c(d).\overline{d}\langle r_0\rangle$ and those appearing in other actions are not identical. Remember the scope of d in $c(d)$ is limited to only actions guarded by it.

$$
\begin{aligned}
(\nu\, r_0, r_1, r_2)(!\,(c(d).\overline{d}\langle r_0\rangle \\
&\mid r_0(x).([x{=}\mathtt{f}]\overline{d}\langle r_1\rangle + [x{=}\mathtt{g}]\overline{d}\langle r_1\rangle) \\
&\mid r_1(y).[y{=}\mathtt{h}]\overline{d}\langle r_2\rangle \\
&\mid \overline{r_2}\langle \mathtt{a}\rangle))
\end{aligned}
$$

To make each processes completely independent, we have to ensure that each process should receive a channel, f, to return the access.

$$
\begin{aligned}
(\nu\, r_0, r_1, r_2)(!\,(c(d).\overline{d}\langle r_0\rangle \\
&\mid r_0(x, f).([x{=}\mathtt{f}]\overline{f}\langle r_1\rangle + [x{=}\mathtt{g}]\overline{f}\langle r_1\rangle) \\
&\mid r_1(y, f).[y{=}\mathtt{h}]\overline{f}\langle r_2\rangle \\
&\mid \overline{r_2}\langle \mathtt{a}\rangle))
\end{aligned}
$$

The encoding however violates the typing principle as r_2 returns only a value, a, while other channels, r_0 and r_1, receive a name and a channel. To preserve the typing scheme, we ensure the value should be returned through a different sort of channel, s. Given the change, the system should return the channel, s, through a different channel other than f. Let v be such a channel to return s, the access to the value. We modify then our encoding to the following.

$$
\begin{aligned}
(\nu\, r_0, r_1, s)(!\,(c(d).\overline{d}\langle r_0\rangle \\
&\mid r_0(x, f, v).([x{=}\mathtt{f}]\overline{f}\langle r_1\rangle + [x{=}\mathtt{g}]\overline{f}\langle r_1\rangle) \\
&\mid r_1(y, f, v).[y{=}\mathtt{h}]\overline{v}\langle s\rangle \\
&\mid \overline{s}\langle \mathtt{a}\rangle))
\end{aligned}
$$

The system first returns r_0 when it receives a request for access to its feature information. Through the channel, other processes can interrogate it by emitting a particular feature name and two channels, f and v, which are used to receive another interrogable channel or a channel to extract a value, respectively. The system returns s through v when it is interrogated through r_1 given a feature name, h. When the system returns a channel through v, not f, it means that now it is ready to offer a value rather than returning another interrogative channel. Once the channel, s, is received, the interrogating processes can extract the value, a, through it.

5.2.5 Feature structures as a set of processes

In what follows, we opt for encoding feature descriptions as is sketched in the above (§5.2.4). To refine it further, we make each process replicable as the replication operator, !, distributes over the parallel composition operator, $\mid$, i.e., $!\,(P \mid Q) \equiv\, !\,P \mid !\,Q$. The above encoding is therefore equivalent to the below.

$$
\begin{aligned}
(\nu\, r_0, r_1, s)(\ !\,c(d).\overline{d}\langle r_0\rangle \\
&\mid\ !\,r_0(x, f, v).([x{=}\mathtt{f}]\overline{f}\langle r_1\rangle + [x{=}\mathtt{g}]\overline{f}\langle r_1\rangle) \\
&\mid\ !\,r_1(y, f, v).[y{=}\mathtt{h}]\overline{v}\langle s\rangle \\
&\mid\ !\,\overline{s}\langle \mathtt{a}\rangle)
\end{aligned}
$$

Since we have introduced a different type of channel, s, to return the value, the feature description (5.6) should also be modified to (5.8).

$$
\boxed{r_0}\begin{bmatrix} f\!: \boxed{r_1}\begin{bmatrix} h\!: \boxed{s}\, a \end{bmatrix} \\ g\!: \boxed{r_1} \end{bmatrix} \qquad (5.8)
$$

To sum up, if we encode the description (5.5), whose channels, r_1, r_2, r_4 and r_5, are replaced by s_1, s_2, s_4, and s_5, respectively, it may be encoded as is shown in the table 5.1. It will be explained in §5.3.4 how items of information can be retrieved from other processes. In the following, we often suppress the processes encoding feature structures, e.g., $FS(c)$ for the feature description, where c indicates the channel through which the information is accessible.

$$
\begin{aligned}
(\nu\, r_0, r_3, s_1, s_2, s_4, s_5)(\ !\,c(d).\overline{d}\langle r_0\rangle \\
&\mid\ !\,r_0(x, f, v).([x{=}\mathtt{cat}]\overline{v}\langle s_1\rangle \\
&\qquad + [x{=}\mathtt{phon}]\overline{v}\langle s_2\rangle \\
&\qquad + [x{=}\mathtt{agr}]\overline{f}\langle r_3\rangle\ ) \\
&\mid\ !\,r_3(y, f, v).([y{=}\mathtt{per}]\overline{v}\langle s_4\rangle \\
&\qquad + [y{=}\mathtt{num}]\overline{v}\langle s_5\rangle) \\
&\mid\ !\,\overline{s_1}\langle \mathtt{verb}\rangle \\
&\mid\ !\,\overline{s_2}\langle \mathtt{walks}\rangle \\
&\mid\ !\,\overline{s_4}\langle \mathtt{3rd}\rangle \\
&\mid\ !\,\overline{s_5}\langle \mathtt{sing}\rangle\ )
\end{aligned}
$$

Table 5.1. Encoding the lexicon for *"walk"* (5.5) as a set of processes

5.3 Parsing as evolution of systems

5.3.1 Agents

We explain how a sentence can be parsed as the result of interactions between processes. Suppose P, Q, and R are subsystems encoding features of 'a', 'man', and 'walk', respectively. Although we have previously depicted the whole system for the sentence as if each subsystems were connected directly with each other (Figure 5.1), they are not connected in that way, but through independent processes accompanying to each subsystem. Let C_i be such an accompanying process. The relation between them and with subsystems encoding features can be depicted as is shown in the figure 5.2.

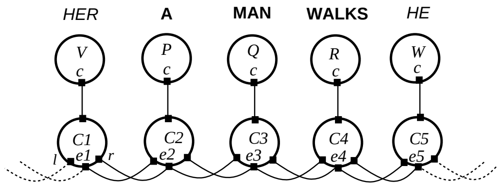


Figure 5.2. The subsystems connected via accompanying processes

In the figure, for example, $C3$ acts as an agent for Q encoding the lexical entry, 'man', and is connected to it through a private channel, c. With other agents $C2$ and $C4$ acting for P and R, respectively, it is connected at left and righthand side. $C3$ offers the access to itself at e_3, which is open only to $C2$ and $C4$. This is done by informing only $C2$ and $C4$ of the access channel, e_3.

5.3.2 Agents to trigger actions of other agents

To start the computation, each agent will trigger actions of its neighbours by emitting to them its channel, e_i, through which the neighbour may communicate with it. Suppose the feature descriptions of 'a' and 'man' are abbreviated to $FS2(c)$ and $FS3(c)$, respectively. The agents $C2$ and $C3$ with P and Q encoding these feature descriptions, are encoded as subsystems as is shown in the table 5.2. The e_2 and e_3 appearing in the definitions of $C2$ and $C3$ are the same although the definition does not mean it.[4] The definitions for $C2_l(y)$, $C2_r(z)$, $C3_l(y)$, and $C3_r(z)$ will be given later.

An agent may communicate with another agent when it receives the access, e.g., e_2 or e_3, through a particular channel, l (left) or r (right). Let us see how the system of $(C2 \mid C3)$ would evolve into another state. The table 5.3 shows the steps until $C3_l(y)$ is triggered

[4] This is because for simplicity we omit the process by which $C2$ and $C3$ are generated given the same bound names, e_2 and e_3. $C2$ is actually parameterised over e_1, e_2, and e_3, and $C3$ over e_2, e_3, and e_4. They should look like $C2(x_2, y_2, z_2)$ and $C3(x_3, y_3, z_3)$, respectively, and the parameters x_2, y_2, and z_2 will be substituted by e_1, e_2, and e_3, and x_3, y_3, and z_3 by e_2, e_3, and e_4. The names from e_1 to e_4 are all bound names to be created by the process initialising $C2$ and $C3$.

$$
\begin{array}{lll}
C2 =_{\mathrm{def}} (\nu\, e_1, e_2, e_3)(& & \\
 & & (\nu\, c, l, r)(\,!\,\overline{e_2}\langle c, l, r\rangle \\
 & & \mid FS2(c) \\
 & & \mid\ !\,l(y).C2_l(y) \\
 & & \mid\ !\,r(z).C2_r(z)) \\
 & \mid e_1(c_1, l_1, r_1).\overline{r_1}\langle e_2\rangle & \\
 & \mid e_3(c_3, l_3, r_3).\overline{l_3}\langle e_2\rangle\ ) &
\end{array}
$$

$$
\begin{array}{lll}
C3 =_{\mathrm{def}} (\nu\, e_2, e_3, e_4)(& & \\
 & & (\nu\, c, l, r)(\,!\,\overline{e_3}\langle c, l, r\rangle \\
 & & \mid FS3(c) \\
 & & \mid\ !\,l(y).C3_l(y) \\
 & & \mid\ !\,r(z).C3_r(z)) \\
 & \mid e_2(c_2, l_2, r_2).\overline{r_2}\langle e_3\rangle & \\
 & \mid e_4(c_4, l_4, r_4).\overline{l_4}\langle e_3\rangle\ ) &
\end{array}
$$

Table 5.2. The definition of $C2$ and $C3$

with the access to e_2. We suppress in the table $e_1(c_1, l_1, r_1).\overline{r}\langle e_2\rangle$ and $e_4(c_4, l_4, r_4).\overline{l}\langle e_3\rangle$ for simplicity. At the second step, $C3$ creates a copy of $\overline{e_3}\langle c, l, r\rangle$. Then, it offers the process $e_3(c_3, l_3, r_3).\overline{l_3}\langle e_2\rangle$ items of information through e_3. At the third step, the channel l of $C3$ is emitted, and in turn through which e_2, the access to $C2$, is returned by executing the copy, $l(y).C3_l(y)$. Through similar steps, $C2_r(z)$ will be trigerred with the access to e_3.

5.3.3 Recording and retrieving access to neighbours

The process, $C3_l(y)$, first records the access to its left neighbour, e_2. We record the information on left and right neighbours at particular locations, pln_3 and prn_3, abbreviation for Private record of Left Neighbour and Private record of Right Neighbour, respectively. The output action, $\overline{e_3}\langle c, l, r\rangle$, is modified to $\overline{e_3}\langle c, l, r, ln_3, rn_3\rangle$ by adding to it ln_3 and rn_3, both of which are private to $C3$. Other processes can get access to the locations, ln_3 and rn_3, to retrieve the access to $C3$'s left and right neighbours. Let $C3'$ be the modified version of $C3$. The channels, ln_3 and rn_3, are private to $C3'$. After $C3_l(y)$ and $C3_r(z)$ record their neighbours, the whole system should turn into the one shown in the table 5.4.

The last two processes, $!\,\overline{pln_3}\langle e_2\rangle$ and $!\,\overline{prn_3}\langle e_4\rangle$, are created by $C3_l(y)$ and $C3_r(z)$, respectively. With these replicable processes, the other two processes, $ln_3(x).pln_3(y).\overline{x}\langle y\rangle$

1. The initial state of $(\nu\, e_2, e_3)(C2 \mid C3)$

$$
\begin{aligned}
&(\nu\, e_2, e_3)((\nu\, c, l, r)(\, !\overline{e_2}\langle c, l, r\rangle \mid FS2(c)\\
&\qquad \mid !\, l(y).C2_l(y) \mid !\, r(z).C2_r(z)) \mid e_3(c_3, l_3, r_3).\overline{l_3}\langle e_2\rangle\\
&\qquad \mid (\nu\, c, l, r)(!\overline{e_3}\langle c, l, r\rangle \mid FS3(c)\\
&\qquad \mid !\, l(y).C3_l(y) \mid !\, r(z).C3_r(z)) \mid e_2(c_2, l_2, r_2).\overline{r_2}\langle e_3\rangle)
\end{aligned}
$$

2. $C3$ creates a copy of $\overline{e_3}\langle c, l, r\rangle$

$$
\begin{aligned}
&(\nu\, e_2, e_3)((\nu\, c, l, r)(\, !\overline{e_2}\langle c, l, r\rangle \mid FS2(c)\\
&\qquad \mid !\, l(y).C2_l(y) \mid !\, r(z).C2_r(z)) \mid e_3(c_3, l_3, r_3).\overline{l_3}\langle e_2\rangle\\
&\qquad \mid (\nu\, c, l, r)(\overline{e_3}\langle c, l, r\rangle \mid !\overline{e_3}\langle c, l, r\rangle \mid FS3(c)\\
&\qquad \mid !\, l(y).C3_l(y) \mid !\, r(z).C3_r(z)) \mid e_2(c_2, l_2, r_2).\overline{r_2}\langle e_3\rangle)
\end{aligned}
$$

3. The process $e_3(c_3, l_3, r_3).\overline{l_3}\langle e_2\rangle$ interacts with the created process. The scope of l of $C3$ extrudes to $C2$.

$$
\begin{aligned}
&(\nu\, e_2, e_3)((\nu\, c, l, r)(\, !\overline{e_2}\langle c, l, r\rangle \mid FS2(c)\\
&\qquad \mid !\, l(y).C2_l(y) \mid !\, r(z).C2_r(z)) \mid (\nu\, l)(\, \overline{l}\langle e_2\rangle\\
&\qquad \mid (\nu\, c, r)(\overline{e_3}\langle c, l, r\rangle \mid !\overline{e_3}\langle c, l, r\rangle \mid FS3(c)\\
&\qquad \mid !\, l(y).C3_l(y) \mid !\, r(z).C3_r(z))) \mid e_2(c_2, l_2, r_2).\overline{r_2}\langle e_3\rangle)
\end{aligned}
$$

4. $C3$ creates a copy of $l(y).C3_l(y)$

$$
\begin{aligned}
&(\nu\, e_2, e_3)((\nu\, c, l, r)(\, !\overline{e_2}\langle c, l, r\rangle \mid FS2(c)\\
&\qquad \mid !\, l(y).C2_l(y) \mid !\, r(z).C2_r(z)) \mid (\nu\, l)(\, \overline{l}\langle e_2\rangle\\
&\qquad \mid (\nu\, c, r)(\overline{e_3}\langle c, l, r\rangle \mid !\overline{e_3}\langle c, l, r\rangle \mid FS3(c)\\
&\qquad \mid l(y).C3_l(y) \mid !\, l(y).C3_l(y) \mid !\, r(z).C3_r(z))) \mid e_2(c_2, l_2, r_2).\overline{r_2}\langle e_3\rangle)
\end{aligned}
$$

5. Through l, the created process receives e_2, and the process $C3_l(y)$ becomes active. The scope of l of $C3$ migrates to itself.

$$
\begin{aligned}
&(\nu\, e_2, e_3)((\nu\, c, l, r)(\, !\overline{e_2}\langle c, l, r\rangle \mid FS2(c)\\
&\qquad \mid !\, l(y).C2_l(y) \mid !\, r(z).C2_r(z))\\
&\qquad \mid (\nu\, c, l, r)(\overline{e_3}\langle c, l, r\rangle \mid !\overline{e_3}\langle c, l, r\rangle \mid FS3(c)\\
&\qquad \mid C3_l(e_2)\{^{e_2}/_y\} \mid !\, l(y).C3_l(y) \mid !\, r(z).C3_r(z))) \mid e_2(c_2, l_2, r_2).\overline{r_2}\langle e_3\rangle)
\end{aligned}
$$

Table 5.3. The steps until $C3_l(y)$ is triggered

$$
\begin{aligned}
C3 =_{\mathrm{def}}\ &(\nu\, e_2, e_3, e_4)(\\
&\quad (\nu\, c, l, r, ln_3, rn_3, pln_3, prn_3)(\\
&\quad !\overline{e_3}\langle c, l, r, ln_3, rn_3\rangle\\
&\quad \mid FS3(c)\\
&\quad \mid !\, l(y).C3_l(y)\\
&\quad \mid !\, r(z).C3_r(z)\\
&\quad \mid !\, ln_3(x).pln_3(y).\overline{x}\langle y\rangle\\
&\quad \mid !\, rn_3(x).prn_3(y).\overline{x}\langle y\rangle\\
&\quad \mid !\, \overline{pln_3}\langle e_2\rangle\\
&\quad \mid !\, \overline{prn_3}\langle e_4\rangle\ ))
\end{aligned}
$$

Table 5.4. The state of $C3'$ after recording neighbours

and $rn_3(x).prn_3(y).\overline{x}\langle y\rangle$, can return the access to $C2$ and $C4$, e_2 and e_4, through the received channel x when they receive it through ln_3 and rn_3, respectively.

The two processes, $C3_l(y)$ and $C3_r(z)$, are defined as below. The names, y and z, are received through $l(y)$ and $r(z)$, respectively. Thus $C3_l(y)$ can create the process, $!\overline{pln_3}\langle e_2\rangle$, because y is substituted by e_2.

$$C3_l(y) =_{\mathrm{def}}\ !\,\overline{pln_3}\langle y\rangle$$

$$C3_r(z) =_{\mathrm{def}}\ !\,\overline{prn_3}\langle z\rangle$$

Although we assume here that there exists only one neighbour for each position, at left and right, which means that $C3_l(y)$ and $C3_r(z)$ are assumed to be executed only once, the assumption will be relaxed and the interaction for recording and retrieving the information on neighbours becomes more complex by employing list data structure. The development is necessary to parse every possible syntactic structures. We will present our solution in §5.4.

5.3.4 Retrieving feature information

When trigerred, the process will look into the feature information through the given channel and may generate a new agent depending on the items of information it extracts of. Suppose that the process, $C3_l(e_2)\{^{e_2}/_y\}$, interacts with $C2$ through e_2 and generates a new agent corresponding to noun phrase if it can extract items of information that indicates the category of its left neighbour is determiner. Let $C3_l(y)'$ be a process performing the series of actions. The definition of $C3_l(y)$ is modified, added the process as follows:

$$C3_l(y) =_{\mathrm{def}}\ !\,\overline{pln_3}\langle y\rangle \mid C3_l(y)'$$

where $C3_l(y)'$ is defined as

$$C3_l(y)' =_{\text{def}} (\nu\, d, f, v)(y(c, l, r, ln, rn).\overline{c}\langle d\rangle.d(r_0).\overline{r_0}\langle \texttt{cat}, f, v\rangle.v(s_1).s_1(x).[x = \texttt{det}]C3_{lg})$$

The table 5.5 shows how it retrieves the information on category of $C2$. The process performs the following series of actions:

(a) $e_2(c, l, r, ln, rn)$: The process receives c, l, r, ln, and rn through e_2, which substituted y.

(b) $\overline{c}\langle d\rangle$: It emits its private channel d through c.

(c) $d(r_0)$: Then it waits for the reply at the channel, which will replace the channel to the top level of the feature structure for r_0.

(d) $\overline{r_0}\langle \texttt{cat}, f, v\rangle$: Once it gets the channel, it emits a request, `cat`, along with two channels, f and v. The second channel, v, is used to get access to the information on category.

(e) $v(s_1)$: The reply is received through v to substitute s_1.

(f) $s_1(x)$: Then it can retrieve the information on category through s_1.

(g) $C3_{lg}$: It will proceed to next actions of generating a subsystem corresponding to noun phrase if the category matches to `det`. It terminates otherwise.

5.3.5 Generating new agents

We look into how a new agent can be generated by considering as an example what processes $C3_{lg}$ above may generate. Let $C6$ be the subsystem to be generated and $FS6(c)$ the set of processes encoding its feature information. The figure 5.3 shows its relation with other agents. Its left neighbour is $C1$ and right one is $C4$. Analogous to $C2$ and $C3$, the subsystem could be defined as is shown in the table 5.6.

To generate such a subsystem, $C3_{lg}$ will evolve into $C6(left, self, right)$, where the parameters, $left, self$, and $right$, must be instantiated with channels to its left neighbour, itself, and its right neighbour, respectively. The channel, $self$, can be instantiated with an arbitrary channel created freshly, say e_6, but $left$ and $right$ have to be instantiated with e_1 and e_4. Let $C3'_{lg}$ be the process retrieving these channels, defined at the moment as

$$C3_{lg} =_{\text{def}} (\nu\, e_6)(C3'_{lg}.C6(left, e_6, right))$$

It is not hard for $C3$ to retrieve e_4 as it is its right neighbour. It needs only to perform once the input action, $prn_3(right)$. Let $C3^r_{lg}$ be the process to retrieve the access to the right neighbour, which is defined as

The step 1

$$\begin{array}{l}(\nu\, c, l, r, ln_2, rn_2)(\overline{e_2}\langle c, l, r, ln_2, rn_2\rangle\\ \mid (\nu\, r_0, s_1)(\ c(d).\overline{d}\langle r_0\rangle \mid r_0(x, f, v).[x{=}\texttt{cat}]\overline{v}\langle s_1\rangle \mid \overline{s_1}\langle \texttt{det}\rangle))\\ \mid (\nu\, d, f, v)(e_2(c, l, r, ln, rn).\overline{c}\langle d\rangle.d(r_0).\ \overline{r_0}\langle \texttt{cat}, f, v\rangle.v(s_1).s_1(x).\ [x = \texttt{det}]C3_{lg})\end{array}$$

The step 2

$$\begin{array}{l}(\nu\, c)((\nu\, r_0, s_1)(\ c(d).\overline{d}\langle r_0\rangle \mid r_0(x, f, v).[x{=}\texttt{cat}]\overline{v}\langle s_1\rangle \mid \overline{s_1}\langle \texttt{det}\rangle)\\ \mid (\nu\, d, f, v)(\overline{c}\langle d\rangle.d(r_0).\ \overline{r_0}\langle \texttt{cat}, f, v\rangle.v(s_1).s_1(x).\ [x = \texttt{det}]C3_{lg}))\end{array}$$

The step 3

$$\begin{array}{l}(\nu\, d)((\nu\, r_0, s_1)(\ \overline{d}\langle r_0\rangle \mid r_0(x, f, v).[x{=}\texttt{cat}]\overline{v}\langle s_1\rangle \mid \overline{s_1}\langle \texttt{det}\rangle)\\ \mid (\nu\, f, v)(d(r_0).\ \overline{r_0}\langle \texttt{cat}, f, v\rangle.v(s_1).s_1(x).\ [x = \texttt{det}]C3_{lg}))\end{array}$$

The step 4

$$\begin{array}{l}(\nu\, r_0)((\nu\, s_1)(\ r_0(x, f, v).[x{=}\texttt{cat}]\overline{v}\langle s_1\rangle \mid \overline{s_1}\langle \texttt{det}\rangle)\\ \mid (\nu\, f, v)(\overline{r_0}\langle \texttt{cat}, f, v\rangle.v(s_1).s_1(x).\ [x = \texttt{det}]C3_{lg}))\end{array}$$

The step 5

$$\begin{array}{l}(\nu\, v)((\nu\, s_1)(\ \overline{v}\langle s_1\rangle \mid \overline{s_1}\langle \texttt{det}\rangle)\\ \mid v(s_1).s_1(x).\ [x = \texttt{det}]C3_{lg}\ )\end{array}$$

The step 6

$$\begin{array}{l}(\nu\, s_1)(\ \overline{s_1}\langle \texttt{det}\rangle\\ \mid s_1(x).\ [x = \texttt{det}]C3_{lg}\ )\end{array}$$

The step 7

$$C3_{lg}$$

Table 5.5. Retrieving the information on category of $C2$

$$
\begin{aligned}
C6(left, self, right) =_{\text{def}} & \ (\nu\, c, l, r, ln_6, rn_6, pln_6, prn_6)(\\
& \ !\,\overline{self}\langle c, l, r, ln_6, rn_6\rangle \\
& \mid FS6(c) \\
& \mid \,!\,l(y).C6_l \\
& \mid \,!\,r(z).C6_r \\
& \mid \,!\,ln_6(x).pln_6(y).\overline{x}\langle y\rangle \\
& \mid \,!\,rn_6(x).prn_6(y).\overline{x}\langle y\rangle \\
& \mid left(c, l, r, ln, rn).\overline{r}\langle self\rangle \\
& \mid right(c, l, r, ln, rn).\overline{l}\langle self\rangle\ )
\end{aligned}
$$

Table 5.6. The definition of $C6$

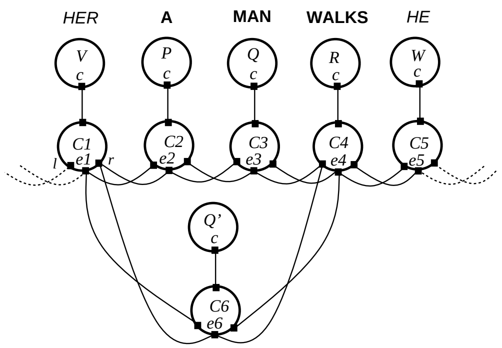


Figure 5.3. The agent C6 and its relation with others

$$C3^r_{lg} =_{\text{def}} \ prn_3(right)$$

It is however more difficult to retrieve e_1, the access to $C1$, as $C3$ does not have direct access to it. To retrieve it, it has to consult $C2$. Let $C3^l_{lg}$ be the process to retrieve e_1, which is defined as follows:

$$C3^l_{lg} =_{\text{def}} \ (\nu\, w)(pln_3(v).v(c, l, ln, rn).\overline{ln}\langle w\rangle.w(left))$$

The first action, $pln_3(v)$, substitutes e_2 for v, as there is a replicable process, $!\,\overline{pln_3}\langle e_2\rangle$, recording the access to the left neighbour of $C3$. Then, by executing $e_2(c, l, ln, rn)$, it can retrieve ln, the access to $C2$, to retrieve the access to its left neighbour. Through the channel, it emits a private channel, w, to get the access to $C2$'s left neighbour, e_1.

The table (5.7) shows how $left$ and $right$ in $C6(left, self, right)$ will be substituted by e_1 and e_4, respectively. The first line shows the relevant definition of $C2$ to return the access to its left neighbour. The process, $\overline{pln_2}\langle e_1\rangle$, offers the access through pln_2. The second line shows processes offering the access to left and right neighbours of $C3$. The third line shows the sequentially combined process of $C3^r_{lg}$ and $C3^l_{lg}$. The table can be explained as follows:

(a) The channel, $right$, in $C6(left, e_6, right)$ is substituted by e_4 when $prn_3(right)$ interacts with $\overline{prn_3}\langle e_4\rangle$.

(b) v in $v(c, l, ln, rn)$ is substituted by e_2 when $pln_3(v)$ interacts with $\overline{pln_3}\langle e_2\rangle$, which makes it possible to retrieve information from e_2.

(c) After retrieving information from e_2, the channel ln_2 replaces ln in $\overline{ln}\langle w\rangle$.

(d) Through the channel, the private channel w is emitted to replace x in $ln_2(x)$. The scope of w extrudes to $C2$.

(e) y is replaced by e_1 upon the interaction between $pln_2(y)$ and $\overline{pln_2}\langle e_1\rangle$.

(f) Finally, it gets e_1 through w to replace it for $left$.

Now we have almost done with the generation procedure. At last, $C3$ needs to inform $C6$ that $C1$ and $C4$ are its neighbours because the actions, $e_1(c, l, r).\overline{r}\langle e_6\rangle$ and $e_4(c, l, r).\overline{l}\langle e_6\rangle$, the instantiated actions of $left(c, l, r, ln, rn).\overline{r}\langle self\rangle$ and $right(c, l, r, ln, rn).\overline{l}\langle self\rangle$, only inform $C1$ and $C4$ that $C6$ is newly created and positioned at right to $C1$ and at left to $C4$, respectively. The notification can be done by the following two processes:

$$e_6(c, l, r, ln, rn).\overline{r}\langle e_4\rangle$$
$$e_6(c, l, r, ln, rn).\overline{l}\langle e_1\rangle$$

which leads to creating $!\,\overline{prn_6}\langle e_4\rangle$ and $!\,\overline{pln_6}\langle e_1\rangle$, respectively, and possibly to another generation process. Now that we have already got the access, e_1 and e_4, these processes

The initial state

$$\begin{aligned}&(\nu\, c, l, r, ln_2, rn_2, pln_2)(\overline{e_2}\langle c, l, r, ln_2, rn_2\rangle.ln_2(x).pln_2(y).\overline{x}\langle y\rangle \mid \overline{pln_2}\langle e_1\rangle)\\ &\mid (\nu\, pln_3, prn_3)(\overline{pln_3}\langle e_2\rangle \mid \overline{prn_3}\langle e_4\rangle\\ &\mid prn_3(right).(\nu\, w)(pln_3(v).v(c, l, ln, rn).\overline{ln}\langle w\rangle.w(left).C6(left, e_6, right)))\end{aligned}$$

The step 1

$$\begin{aligned}&(\nu\, c, l, r, ln_2, rn_2, pln_2)(\overline{e_2}\langle c, l, r, ln_2, rn_2\rangle.ln_2(x).pln_2(y).\overline{x}\langle y\rangle \mid \overline{pln_2}\langle e_1\rangle)\\ &\mid (\nu\, pln_3)(\overline{pln_3}\langle e_2\rangle\\ &\mid (\nu\, w)(pln_3(v).v(c, l, ln, rn).\overline{ln}\langle w\rangle.w(left).C6(left, e_6, e_4)))\end{aligned}$$

The step 2

$$\begin{aligned}&(\nu\, c, l, r, ln_2, rn_2, pln_2)(\overline{e_2}\langle c, l, r, ln_2, rn_2\rangle.ln_2(x).pln_2(y).\overline{x}\langle y\rangle \mid \overline{pln_2}\langle e_1\rangle)\\ &\mid (\nu\, w)(e_2(c, l, ln, rn).\overline{ln}\langle w\rangle.w(left).C6(left, e_6, e_4))\end{aligned}$$

The step 3

$$\begin{aligned}&(\nu\, ln_2)((\nu\, pln_2)(ln_2(x).pln_2(y).\overline{x}\langle y\rangle \mid \overline{pln_2}\langle e_1\rangle)\\ &\mid (\nu\, w)(\overline{ln_2}\langle w\rangle.w(left).C6(left, e_6, e_4)))\end{aligned}$$

The step 4

$$\begin{aligned}&(\nu\, w)((\nu\, pln_2)(pln_2(y).\overline{w}\langle y\rangle \mid \overline{pln_2}\langle e_1\rangle)\\ &\mid w(left).C6(left, e_6, e_4))\end{aligned}$$

The step 5

$$\begin{aligned}&(\nu\, w)(\overline{w}\langle e_1\rangle\\ &\mid w(left).C6(left, e_6, e_4))\end{aligned}$$

The step 6

$$C6(e_1, e_6, e_4)$$

Table 5.7. Retrieving the access to $C1$ and $C4$

can be initialised at the same time as C6 when x and y are replaced by them. The whole process of generation can be summarised as

$$\begin{aligned}C3_{lg} =_{\text{def}}\ &(\nu\, e_6)\ (C3^r_{lg}.C3^l_{lg}.(C6(left, e_6, right)\\ &\mid e_6(c, l, r, ln, rn).\overline{r}\langle right\rangle \mid e_6(c, l, r, ln, rn).\overline{l}\langle left\rangle))\end{aligned}$$

Through similar steps, the sentence *"a man walks"* can be analysed to be a sentence. When $C6(left, self, right)$ is instantiated with e_1, e_6, and e_4, it is accompanied by $e_4(c, l, r, ln, rn).\overline{l}\langle e_6\rangle$, which informs $C4$ that a new left neighbour is created at left with the access, e_6. $C4$ will then check through the channel if the category of $C6$ matches to `np` to generate another new agent $C7$ corresponding to sentence category if it is the case.

Note also the information on features can be retrieved from agents in long distance, not limited to just neighbours. As is shown in the above, the agent $C3$ could retrieve e_1, the access to $C1$, with which it could extract information of $C1$ crossing over $C2$. Nothing prohibits an agent from searching items of information in long distance no matter how far the source is away as long as it is reachable through channels. The number of sources is also unlimited. Although in the above example $C3$ looks into only $C2$, it could look into other agents as well. Our parser can therefore analyze tree structures, not limited to binary branching.

5.4 A concurrent chart parser

5.4.1 Multiple neighbours

In the above example of *"a man walks"*, we did not worry about how many neighbours might exist for an agent. We assumed that an agent need to communicate with at most one neighbour at either side. The assumption causes no problem as long as we parse a simple sentence such as *"a man walks"*. Let us see the situation by using more conventional diagram. The figure 5.4 shows a chart for the sentence. In this guise, one can conceive of each agent, $C2, C3, C4, C6$, and $C7$ as *edges*. As we have seen, $C3$ needs only to communicate with $C2$ to create $C6$. $C4$ also needs only to communicate with $C6$ to create $C7$. Although $C4$ might have communicated with $C3$ in vain, it causes no problem as it is notified of $C6$ when it is created so that it can look into it.

The situation may be different when we come to parsing sentences of more complex syntax. Let us examine a sentence, *"he sees desks in the room."*, which is conventionally assigned two syntactic trees. One structure can be represented as [sees [desks [in the room]]], where PP 'in the room' is related with NP 'desks'. The other can be represented

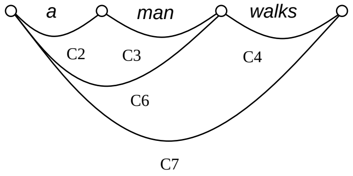


Figure 5.4. A chart of *"a man walks"*

as [[sees desks] [in the room]], where PP is related with VP, 'sees desks'. The figure 5.5 shows its chart. One can see the agent for 'in' has two left neighbours, one for 'desk' and the other for 'sees desks'.

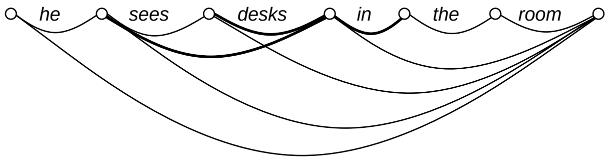


Figure 5.5. A chart of *"he sees desks in the room"*

It is important for the agent $D4$ acting for 'in' to record the access to both agents $D3$ and $D11$, acting for 'desks' and 'sees desks', respectively. Observe what the state the parser in when $D11$ is created before $D7$, the agent for 'in the room' (Figure 5.6). When it is created, $D11$ knows only $D4$ as its right neighbour and have access to it, $f4$. $D4$ is notified of the existence of $D11$ when it is created and knows it has two left neighbours, $D3$ and $D11$, with the access to them, $f3$ and $f11$. These agents, $D3$ and $D11$, wait for $D7$ to be created. When it is created, $D7$ inherits $D4$'s access to left neighbours, $f3$ and $f11$ (Figure5.7). It can therefore notify $D3$ and $D11$, through $f3$ and $f11$, of its creation and the access to itself, $f7$ (Figure 5.8). Given the access, $D3$ can creates $D8$, the agent for NP 'desks in the room', and $D11$ can create $D12$, the agent for VP 'sees desks in the room'.

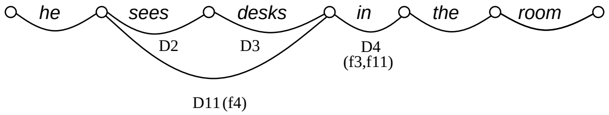


Figure 5.6. The state when $D11$ is created

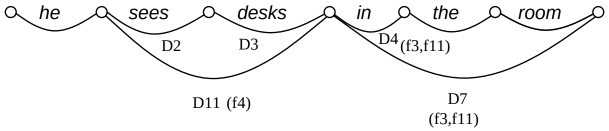


Figure 5.7. The state when $D7$ is created

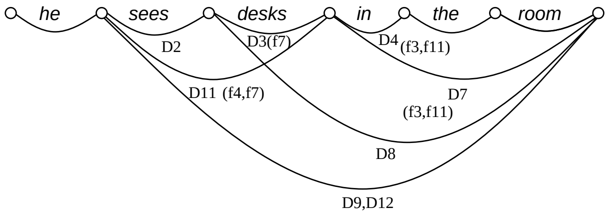


Figure 5.8. The access to $C7$ passed to $D3$ and $D11$

We have to modify our encoding to allow agents to record more than one neighbours. Before proceeding to modification, we look into what problem might happen if we apply the previously presented encoding to the sentence. Assume $D4$ is encoded as is shown in the table 5.8, where $D4_l(y) =_{\mathrm{def}} !\,\overline{pln_4}\langle y\rangle$. When it receives the access to $D3$ and $D11$, i.e., $f3$ and $f11$, the process is evoked to create two processes, $!\,\overline{pln_4}\langle f_3\rangle$ and $!\,\overline{pln_4}\langle f_{11}\rangle$. When $D7$ is created, these channels must be retrieved of $D4$, which is done by interacting with the process, $ln_4(x).pln_4(y).\overline{x}\langle y\rangle$. Now the problem comes here. Since there are two sorts of processes providing it with the access to left neighbour, the retrieving process receives either $f4$ or $f11$ randomly. The process may try to retrieve these channels by repeating the action several times. But there is no guarantee that it succeeds in retrieving both as the chance is completely random.

$$
\begin{aligned}
D4 =_{\mathrm{def}}\ & (\nu\, f_3, f_4, f_5)(\\
& \quad (\nu\, c, l, r, ln_4, rn_4, pln_4, prn_4)(\\
& \qquad\qquad !\,\overline{f_4}\langle c, l, r, ln_4, rn_4\rangle\\
& \qquad\qquad \mid FSd4(c)\\
& \qquad\qquad \mid\ !\,l(y).D4_l(y)\\
& \qquad\qquad \mid\ !\,r(z).D4_r(z)\\
& \qquad\qquad \mid\ !\,ln_4(x).pln_4(y).\overline{x}\langle y\rangle\\
& \qquad\qquad \mid\ !\,rn_4(x).prn_4(y).\overline{x}\langle y\rangle)\\
& \quad \mid f_3(c, l, r, ln, rn).\overline{r}\langle f_4\rangle\\
& \quad \mid f_5(c, l, r, ln, rn).\overline{l}\langle f_4\rangle\ )
\end{aligned}
$$

Table 5.8. The definition of $D4$

5.4.2 Encoding list as a set of processes

To solve the problem, we introduce list data structure to our encoding. If we can record the left neighbours of $D4$ as a list of $\langle f3, f11\rangle$, other processes can retrieve both without relying on luck. Since it has already been investigated in the literature how list structure can be encoded using the π-calculus [Mil93b], we only go though it briefly. The sections from §5.4.2 to §5.4.4 are due to Milner [Mil93b].

The figure 5.9 depicts a single list-cell located at l_0, storing a pointer to its value v and another to its successor cell located at l_1. It may be encoded as

$$l_0(c, n).\overline{c}\langle v, l_1\rangle$$

which receives two names in sequence at l_0 to substitute the first one for c, and emit through it the sequence of v and l_1. By parameterising it over l_0, v and l_1, a parametric process, $Cell$, can be defined as

$$Cell(l_0, v, l_1) =_{\mathrm{def}}\ l_0(c, n).\overline{c}\langle v, l_1\rangle$$

Similarly, the empty list-cell, nil (Figure 5.10), is defined and parameterised over l to define a parametric process, Nil, which returns just $\overline{n}$ without any arguments.

$$Nil(l) =_{\mathrm{def}}\ l(c, n).\overline{n}$$

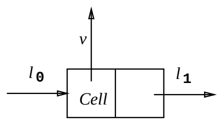


Figure 5.9. A single list-cell

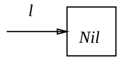


Figure 5.10. An empty list-cell

We consider now how a list, $\langle a, b\rangle$, may be encoded (Figure 5.11). The element a is encoded as a process such that it receives a channel as a request through a particular channel v to return a through it. To encode a and b shown in the figure, one may encode them as follows:

$$A(v) =_{\mathrm{def}}\ v(x).\overline{x}\langle a\rangle$$

$$B(v) =_{\mathrm{def}}\ v(x).\overline{x}\langle b\rangle$$

As a whole, the list is encoded as a system of processes such as

$$
\begin{aligned}
&(\nu\, v_1, l_1)\ (Cell(l_0, v_1, l_1) \mid A(v_1) \mid\\
&\qquad (\nu\, v_2, l_2)\ (Cell(l_1, v_2, l_2) \mid B(v_2) \mid Nil(l_2)))
\end{aligned}
$$

5.4.3 Retrieving values

Now let us see how these values can be retrieved from other processes. We first define $Head$ and $Tail$ as below, which get hold of the first element and the remainder of an arbitrary list value.

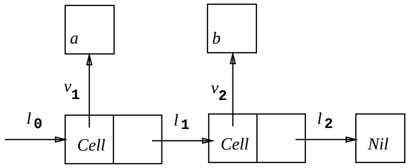


Figure 5.11. A list of a and b

$$Head(l_0, d) =_{\text{def}} (\nu\, c, n)(\overline{l_0}\langle c, n\rangle.c(v, l_1).\overline{x}\langle v\rangle)$$
$$Tail(l_0, e) =_{\text{def}} (\nu\, c, n)(\overline{l_0}\langle c, n\rangle.c(v, l_1).\overline{y}\langle l_1\rangle)$$

$Head$, parameterised over l_0, emits private names, c and n, through l_0 and waits for a sequence of names to be returned through c to substitute v and l_1. The name, v, is the channel to the process storing the first element, and l_1 is another channel to the process encoding the remainder. $Tail$ is almost identical with $Head$, but it returns l_1. To return these channels, both get particular channels, d and e.

The table 5.9 shows the steps for P to retrieve a. The first line of the initial state encodes the list of $\langle a, b\rangle$, and the first process in the second line encodes $Head$ located at l_0, and the second the process P. The steps can be explained as follows:

(a) $Head$ emits c and n through l_0. The scope of these names extrudes to the process encoding the list.

(b) $Head$ receives v_1 and l_1 through c to replace v for v_1.

(c) Through d, P receives v_1 to replace x.

(d) Through v_1, the list process receives w to replace its x.

(e) P receives a through w to replace y.

To retrieve the second element, the retrieving process first receives l_1, then follows the same steps as l_0 is utilised to retrieve a. By repeating the procedure, any elements can be retrieved of list. Finally, to provide for data as many times as required, the process encoding the list must be replicable. That is, the above definition of list should be defined as follows:

$$\begin{aligned}&(\nu\, v_1, l_1)\ (!Cell(l_0, v_1, l_1) \mid\ !A(v_1) \mid \\ &\qquad (\nu\, v_2, l_2)\ (!Cell(l_1, v_2, l_2) \mid\ !B(v_2) \mid\ !Nil(l_2)))\end{aligned}$$

The initial state

$$\begin{aligned}&(\nu\, v_1, l_1)(l_0(c, n).\overline{c}\langle v_1, l_1\rangle \mid v_1(x).\overline{x}\langle a\rangle \mid (\nu\, v_2, l_2)(l_1(c, n).\overline{c}\langle v_2, l_2\rangle \mid v_2(x).\overline{x}\langle b\rangle\ )) \\ &\mid (\nu\, c, n, d)(\overline{l_0}\langle c, n\rangle.c(v, l_1).\overline{d}\langle v\rangle \mid (\nu\, w)(d(x).\overline{x}\langle w\rangle.w(y).P))\end{aligned}$$

Step 1

$$\begin{aligned}&(\nu\, v_1, l_1, c, n)(\overline{c}\langle v_1, l_1\rangle \mid v_1(x).\overline{x}\langle a\rangle \mid (\nu\, v_2, l_2)(l_1(c, n).\overline{c}\langle v_2, l_2\rangle \mid v_2(x).\overline{x}\langle b\rangle\ ) \\ &\mid (\nu\, d)(c(v, l_1).\overline{d}\langle v\rangle \mid (\nu\, w)(d(x).\overline{x}\langle w\rangle.w(y)).P))\end{aligned}$$

Step 2

$$\begin{aligned}&(\nu\, v_1, l_1, c, n)(v_1(x).\overline{x}\langle a\rangle \mid (\nu\, v_2, l_2)(l_1(c, n).\overline{c}\langle v_2, l_2\rangle \mid v_2(x).\overline{x}\langle b\rangle\ ) \\ &\mid (\nu\, d)(\overline{d}\langle v_1\rangle \mid (\nu\, w)(d(x).\overline{x}\langle w\rangle.w(y).P)))\end{aligned}$$

Step 3

$$\begin{aligned}&(\nu\, v_1, l_1, c, n)(v_1(x).\overline{x}\langle a\rangle \mid (\nu\, v_2, l_2)(l_1(c, n).\overline{c}\langle v_2, l_2\rangle \mid v_2(x).\overline{x}\langle b\rangle\ ) \\ &\mid (\nu\, w)(\overline{v_1}\langle w\rangle.w(y).P))\end{aligned}$$

Step 4

$$\begin{aligned}&(\nu\, v_1, l_1, c, n, w)(\overline{w}\langle a\rangle \mid (\nu\, v_2, l_2)(l_1(c, n).\overline{c}\langle v_2, l_2\rangle \mid v_2(x).\overline{x}\langle b\rangle\ ) \\ &\mid w(y).P)\end{aligned}$$

Step 5

$$\begin{aligned}&(\nu\, v_1, l_1, c, n, w)(\ (\nu\, v_2, l_2)(l_1(c, n).\overline{c}\langle v_2, l_2\rangle \mid v_2(x).\overline{x}\langle b\rangle\ ) \\ &\mid P\{{}^{a}/_{y}\})\end{aligned}$$

Table 5.9. Retrieving values of list

5.4.4 Adding values

We also need to add a value to list. To add a value, we place it at the first place. The figure 5.12 depicts how a cell can be concatenated. To connect the process encoding the cell pointing to the new element c and the old one, one can only need to add the following process, defined as Add:

$$Add(l_0, v_3, l_3) =_{\text{def}} \; l_3(c, n).\overline{c}\langle v_3, l_0\rangle$$

By adding it to the system of processes, it will become to return v_3 and l_0 when interrogated at l_3.

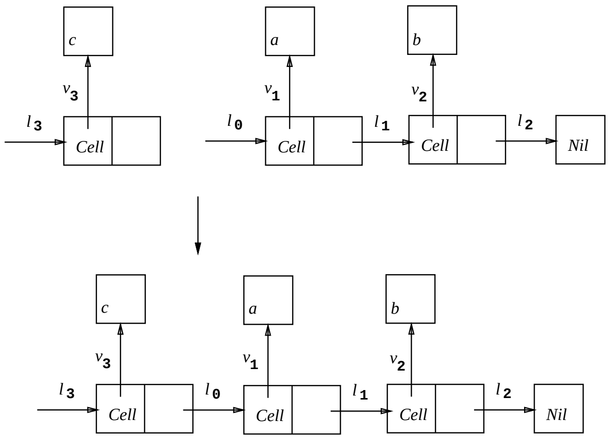


Figure 5.12. Adding c to the list of a and b

It is also possible to add the element at last place, i.e., $\langle a, b, c\rangle$, but it is not worth doing because the order of elements does not matter for us. In fact, a number of operations can be encoded in the π-calculus, let alone retrieving and adding elements. Deleting and inserting an element at a particular location is also possible, but the above encoding is enough for us to record neighbours.

5.4.5 Recording and retrieving neighbours using list

The way to record neighbours is modified by adopting list data structure. The table 5.10 shows the initial state of $C3$, the agent acting for 'man', when list data structure

is applied to. The channels, pln_3 and prn_3, are now used to store the channel, lc_3 (left cell) or rc_3 (right cell), to the first element of the list, not the access to its neighbour. These data is however not replicable. This is because the channel to the first element changes when a new item is added. In the initial state, the list contains only nil item.

The behaviour of $C3_l(y)$ is defined as below. When retrieving the access to its neighbours, it will receive the current channel to the first element of the list through pln_3 to update it to a new channel l that becomes the channel to the new first element. Add creates a new cell and links to it the former first element. The access to its neighbour is stored as value and available upon the interaction with $v(x).\overline{x}\langle y\rangle$. The channels, v and l, are fresh names and private to $C3$.

$$C3_l(y) =_{\text{def}} \; (\nu\, v, l)(pln_3(lc).\overline{pln_3}\langle l\rangle \mid\; !\,Add(lc, v, l) \mid\; !\,v(x).\overline{x}\langle y\rangle)$$

Retrieving the access to neighbours too is modified when we adopt list data structures. $C3$ will now return the channel to the first element, not the access itself. For example, the process $ln_3(x).pln_3(y).\overline{pln_3}\langle y\rangle.\overline{x}\langle y\rangle$ receives a channel through ln_3, retrieves the channel to the first element, and returns it through the received channel. The channel to the first element is restored so that it is not lost by the reading action.

$$
\begin{array}{l}
C3 =_{\text{def}} (\nu\, e_2, e_3, e_4)(\\
\quad (\nu\, c, l, r, ln_3, rn_3, lc_3, rc_3, pln_3, prn_3)(\\
\qquad !\,\overline{e_3}\langle c, l, r, ln_3, rn_3\rangle \\
\qquad FS3(c) \\
\qquad \mid\; !\,l(y).C3_l(y) \\
\qquad \mid\; !\,r(z).C3_r(z) \\
\qquad \mid\; !\,ln_3(x).pln_3(y).\overline{pln_3}\langle y\rangle.\overline{x}\langle y\rangle \\
\qquad \mid\; !\,rn_3(x).prn_3(y).\overline{prn_3}\langle y\rangle.\overline{x}\langle y\rangle \\
\qquad \mid\; \overline{pln_3}\langle lc_3\rangle \;\mid\; !\,Nil(lc_3) \\
\qquad \mid\; \overline{prn_3}\langle rc_3\rangle \;\mid\; !\,Nil(rc_3)\;) \\
\quad \mid\; e_2(c_2, l_2, r_2, ln_2, rn_2).\overline{r_2}\langle e_3\rangle \\
\quad \mid\; e_4(c_4, l_4, r_4, ln_4, rn_4).\overline{l_4}\langle e_3\rangle\;)
\end{array}
$$

Table 5.10. The initial state of $C3$ before recording neighbours

5.4.6 Generating new agents using list

The way that new agents are generated is made complicated by adopting list structure to record neighbours. Formerly, the procedure for $C3$ to generate $C6$ was to read its access to the right neighbour and the left neighbour's access to its left neighbour in order to instantiate $left$ and $right$ with e_1 and e_4, respectively. It also needed to inform itself of its left and right neighbour. The retrieving of access is not much different when we use list data structure. $C3^r_{lg}$ below shows the previous encoding to retrieve its access to the right neighbour, and $C3^{r'}_{lg}$ the modified version. We use x and y to avoid confusion. $C3^l_{lg}$ shows the previous encoding to retrieve the left neighbour's access to its left neighbour, and $C3^{l'}_{lg}$ is its modified version.

$$C3^r_{lg} =_{\text{def}} prn_3(right)$$
$$C3^{r'}_{lg} =_{\text{def}} prn_3(y).\overline{prn_3}\langle y\rangle$$
$$C3^l_{lg} =_{\text{def}} (\nu\, w)(pln_3(v).v(c,l,ln,rn).\overline{ln}\langle w\rangle.w(left))$$
$$C3^{l'}_{lg} =_{\text{def}} (\nu\, w)(pln_3(v).\overline{prn_3}\langle v\rangle.v(c,l,ln,rn).\overline{ln}\langle w\rangle.w(x))$$

The major difference is that the names replacing x and y are not the access to neighbours, but channels to the first element of the list. These channels will be used to initialise $C6$ which is now parameterised over the channels to the first element of the list. Its definition is given in the table 5.11. Note in this encoding, $C6'$ shares the elements of the list locating after lc_6 and rc_6 with $C3$. It does therefore not hold of any processes returning the values by itself in the initial state. One may also notice the processes to inform its left and right neighbours of its creation have disappeared, which should have been such as

$e_1(c_1,l_1,r_1,ln_1,rn_1).\overline{r_1}\langle e_6\rangle$, and

$e_4(c_4,l_4,r_4,ln_4,rn_4).\overline{l_4}\langle e_6\rangle$

The job is taken up by $C3$, the agent creating $C6$, as it is rather complicated job. What else we have to encode are the processes to inform $C6$ of its left and right neighbour. Suppose all these jobs would be performed by $Link(e_6,x,y)$. The definition of $C3_{lg}$ is then modified to $C3'_{lg}$.

$$C3_{lg} =_{\text{def}} (\nu\, e_6)(C3^r_{lg}.C3^l_{lg}.(C6(x,e_6,y) \mid e_6(c,l,r,ln,rn).\overline{r}\langle y\rangle \mid e_6(c,l,r,ln,rn).\overline{l}\langle x\rangle))$$
$$C3'_{lg} =_{\text{def}} (\nu\, e_6)(C3^{r'}_{lg}.C3^{l'}_{lg}.(C6'(x,e_6,y) \mid Link(e_6,x,y)))$$

The definition of $Link(e_6,x,y)$ is given as follows:

- $Link(e_6,x,y) =_{\text{def}} (\nu\, t)(\overline{t}\langle x\rangle.Link_l(e_6,t)) \mid (\nu\, t)(\overline{t}\langle y\rangle.Link_r(e_6,t))$
  The first process does the job concerning left neighbours, and the second concerning right neighbours. Before evoking $Link_l(e_6,t)$, it stores the given channel x at t, which is freshly created and private to it.
- $Link_l(e_6,t) =_{\text{def}} !\,t(x).x(c,n).(c(v,l).\overline{t}\langle l\rangle.Link_{le}(e_6,v)+n)$
  The process repeats a sequence of actions until n (nil) is returned. It retrieves a channel at t, retrieves the cell through it, and evokes $Link_{le}(e_6,v)$ if c is returned. The channel retrievable at t is then updated to l, the channel to the remainder. The process terminates if n is returned.
- $Link_{le}(e_6,v) =_{\text{def}} (\nu\, y)(\overline{v}\langle y\rangle.y(e).(e(c,l,r,ln,rn).\overline{r}\langle e_6\rangle \mid e_6(c,l,r,ln,rn).\overline{l}\langle e\rangle)$
  The process retrieves through y the value e, the access to one of its left neighbours. Then, it informs the neighbour that e_6 is available at its right position and informs $C6$ of its left neighbour.
- $Link_r(e_6,t) =_{\text{def}} !\,t(x).x(c,n).(c(v,l).\overline{t}\langle l\rangle.Link_{re}(e_6,v)+n)$
  $Link_{re}(e_6,v) =_{\text{def}} (\nu\, y)(\overline{v}\langle y\rangle.y(e).(e(c,l,r,ln,rn).\overline{l}\langle e_6\rangle \mid e_6(c,l,r,ln,rn).\overline{r}\langle e\rangle)$
  These do similar jobs as the above ones. $Link_{re}(e_6,v)$ informs the right neighbour that e_6 is available at its left position and informs $C6$ of its right neighbour.

$$\begin{aligned} C6'(lc_6,e_6,rc_6) =_{\text{def}} (\nu\, & c,l,r,ln_6,rn_6,pln_6,prn_6)(\\ & !\,\overline{e_6}\langle c,l,r,ln_6,rn_6\rangle \\ & \mid FS6(c) \\ & \mid !\,l(y).C6_l(y) \\ & \mid !\,r(z).C6_r(z) \\ & \mid !\,ln_6(x).pln_6(y).\overline{pln_6}\langle y\rangle.\overline{x}\langle y\rangle \\ & \mid !\,rn_6(x).prn_6(y).\overline{prn_6}\langle y\rangle.\overline{x}\langle y\rangle \\ & \mid \overline{pln_6}\langle lc_6\rangle \\ & \mid \overline{prn_6}\langle rc_6\rangle\ ) \end{aligned}$$

Table 5.11. The initial state of $C6$ before recording neighbours

5.4.7 Propagating channels to neighbours upward

The parser presented in the above can parse sentences correctly as long as each subsystem corresponding to a word or phrase evolves at different timing. However, a problem arises when subsystems evolve concurrently. Let us see, for example, what trouble we may encounter if $D7$, the agent acting for PP 'in the room', and $D11$, the agent for VP 'sees desks', evolve at the same time.

In the initial state, $D3$, the agent for 'desks' knows its right neighbour is $D4$, and $D4$, the agent for 'in' its left neighbour is $D3$ (5.13). Suppose then $D7$ and $D11$ are created at the same time. Since at that stage $D3$ does not know about $D7$, and also $D4$ does not know about $D11$, the created agents do not know each other (5.14). Subsequently $D3$ and $D4$ will know that $D7$ and $D11$ are created as its neighbours, respectively, but $D11$ and $D7$ will by no means be informed of each other. Therefore, in the next stage only $D8$, the agent for 'desks in the room', is created, and followed by $D9$, the agent for the structure [sees [desks [in the room]]], but $D12$, the agent for the structure [[sees desks] [in the room]] (5.15).

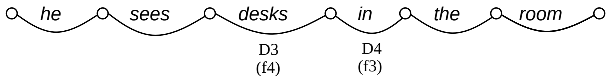


Figure 5.13. The initial state

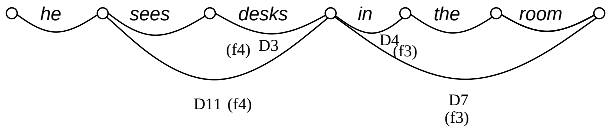


Figure 5.14. The state when $D7$ and $D11$ are created at the same time

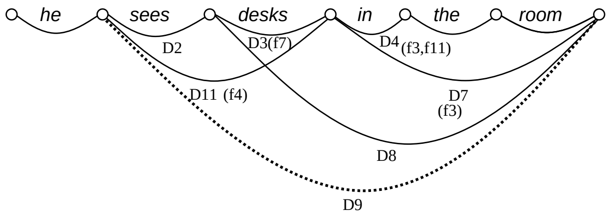


Figure 5.15. The access to $C7$ is lost for $D11$

One way to solve the problem is for the subagent to record with which created agent it shares neighbours and to inform it of the newly created neighbours when it is informed of them. In the example, $D3$ should inform $D11$ that $D7$ is created, and $D4$ should inform $D7$ that $D11$ is created. What each agent should do can be summarised as follows:

- If its information on neighbours, left or right, is used to create an agent, it records

the access to the agent as a list. This is because more than one agent can be created by inheriting the agent's information on its neighbours.

- If it is notified of some agent that is newly created at its neighbouring position, it searches its record to check if the agent is registered. If it is already in the record, the agent does nothing. Otherwise, it informs the agent who shares neighbours of the news. The agent then proceeds to the usual procedure, i.e., adding the channel to the neighbour to its list, and so forth.

The procedure can be encoded as processes by applying the techniques that have been explained in the subsection. We are therefore not bothered to encode it here.

5.5 Discussion

5.5.1 Logical checking of concurrent parsers

The translation from the π-calculus to the combinatorial linear intuitionistic logic enables us to check if the concurrent parser can change its state from a particular one to another as we expect. The figure 5.16 depicts parts of definition of $C2$ and $C3$ in dataflow graph, whose definitions are the basic version presented in §5.3.2. We specify here only the part inside the polygon[5] as logical formulas (Table 5.12).

$a : \overline{e_3}\langle c_3, l_3, r_3\rangle \Rightarrow l_3(y_3) \otimes r_3(z_3)$

$b : l_3(y_3) \Rightarrow \overline{pln_3}\langle y_3\rangle$

$c : r_3(z_3) \Rightarrow \overline{prn_3}\langle z_3\rangle$

$d : e_3(c'_3, l'_3, r'_3) \Rightarrow \overline{l'_3}\langle e_2\rangle$

$e : e_3(c'_3, l'_3, r'_3) \Rightarrow \overline{r'_3}\langle e_4\rangle$

$f : \overline{e_3}\langle c_3, l_3, r_3\rangle \otimes e_3(c'_3, l'_3, r'_3) \Rightarrow !\,(c_3 = c'_3) \otimes !\,(l_3 = l'_3) \otimes !\,(r_3 = r'_3) \otimes l_3(y_3) \otimes \overline{l'_3}\langle e_2\rangle$

$g : \overline{e_3}\langle c_3, l_3, r_3\rangle \otimes e_3(c'_3, l'_3, r'_3) \Rightarrow !\,(c_3 = c'_3) \otimes !\,(l_3 = l'_3) \otimes !\,(r_3 = r'_3) \otimes r_3(z_3) \otimes \overline{r'_3}\langle e_4\rangle$

$h : l_3(y_3) \otimes \overline{l'_3}\langle e_2\rangle \otimes !\,(l_3 = l'_3) \Rightarrow !\,(y_3 = e_2) \otimes \overline{pln_3}\langle y_3\rangle$

$i : r_3(z_3) \otimes \overline{r'_3}\langle e_4\rangle \otimes !\,(r_3 = r'_3) \Rightarrow !\,(z_3 = e_4) \otimes \overline{prn_3}\langle z_3\rangle$

Table 5.12. Logical formulas specifying the particles in the box

Given these formulas, one can infer that after the two interactions occur between $\overline{e_3}\langle c_3, l_3, r_3\rangle$ and $e_3(c'_3, l'_3, r'_3)$, there is a state such that the action $\overline{pln_3}\langle e_2\rangle$ and $\overline{prn_3}\langle e_4\rangle$

[5] Strictly speaking, the node e_4 is outside the polygon, but it is referred to. Also, the node c'_3 is in the polygon, but it is not referred to.

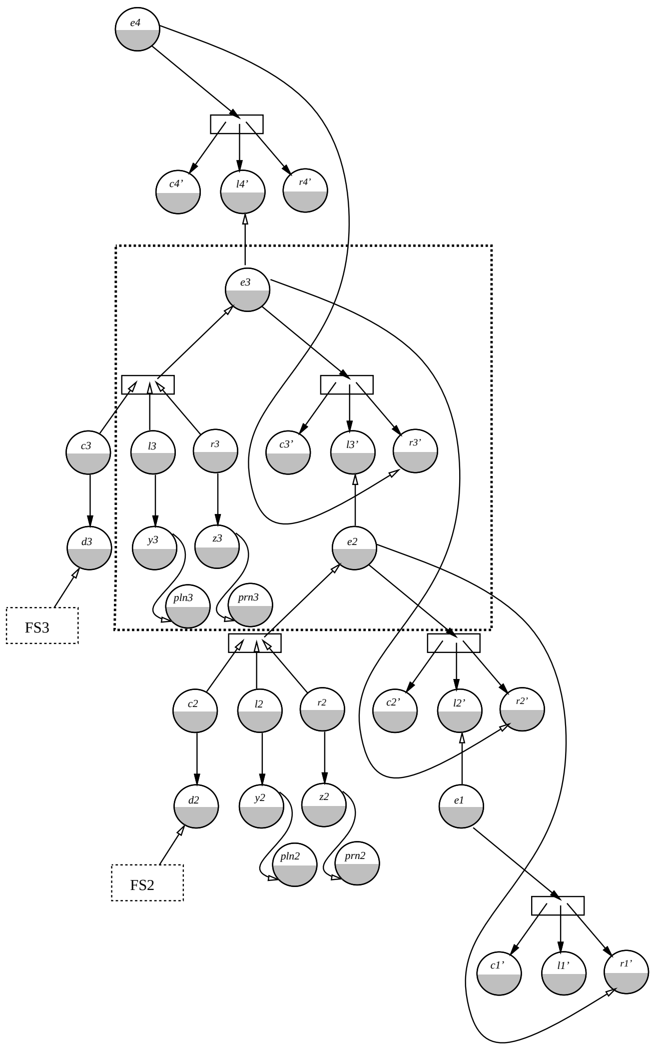


Figure 5.16. The structure of interactions between $C2$ and $C3$

become ready for execution with the substitution environment, $(c_3 = c'_3)$, $(l_3 = l'_3)$, and $(r_3 = r'_3)$, by referring to the special axioms, f, g, h, and i, and by applying to them the rules for parallel and sequential compositions, and substitution. That is, we can construct the following formula:

$$[\overline{e_3}\langle c_3, l_3, r_3\rangle \otimes \overline{e_3}\langle c_3, l_3, r_3\rangle \otimes e_3(c'_3, l'_3, r'_3) \otimes e_3(c'_3, l'_3, r'_3)]$$

$$\Rightarrow \overline{pln_3}\langle e_2\rangle \otimes \overline{prn_3}\langle e_4\rangle \otimes !\,(c_3 = c'_3) \otimes !\,(l_3 = l'_3) \otimes !\,(r_3 = r'_3)$$

5.5.2 Feature description structured based on channel types

In the previous subsection, we have managed to return the discourse marker x in the same way as r_0 is returned. If we encode $C2$ in that way, the full definition may be given as is shown in the table 5.13. The definition seems to be rather chaotic as the result of us encoding them bit by bit. We wonder why we can not treat the whole subsystem as if it were a feature structure.

$$\begin{array}{l}
C2 =_{\text{def}} (\nu\, e_1, e_2, e_3)(\\
\quad (\nu\, c, m, l, r, ln_2, rn_2, lc_2, rc_2, pln_2, prn_2)(\\
\qquad\qquad !\,\overline{e_2}\langle c, m, l, r, ln_2, rn_2\rangle \\
\qquad\qquad |\ FS2(c) \\
\qquad\qquad |\ (\nu\, x)(!\, m(y).\overline{y}\langle x\rangle) \\
\qquad\qquad |\ !\, l(y).C2_l(y) \\
\qquad\qquad |\ !\, r(z).C2_r(z) \\
\qquad\qquad |\ !\, ln_2(x).pln_2(y).\overline{pln_2}\langle y\rangle.\overline{x}\langle y\rangle \\
\qquad\qquad |\ !\, rn_2(x).prn_2(y).\overline{prn_2}\langle y\rangle.\overline{x}\langle y\rangle \\
\qquad\qquad |\ \overline{pln_2}\langle lc_2\rangle \qquad |\ !\, Nil(lc_2) \\
\qquad\qquad |\ \overline{prn_2}\langle rc_2\rangle \qquad |\ !\, Nil(rc_2)\ ) \\
\quad |\ e_1(c_1, l_1, r_1, ln_1, rn_1).\overline{r_1}\langle e_2\rangle \\
\quad |\ e_3(c_3, l_3, r_3, ln_3, rn_3).\overline{l_3}\langle e_2\rangle\ )
\end{array}$$

Table 5.13. The definition of $C2$ to store a discourse marker x

The feature description (5.9) shows a proposal to represent all items of information in the form of feature structure. The channel to the top level is e_2, the access to $C2$. The feature *syn* stores lexical information, *idx* the discourse marker. The feature *add:left* stores the channel to inform $C2$ of its left neighbour and *add:right* the channel to inform it of its right neighbour. The feature *nbh:left* stores the list of left neighbours of $C2$ and *nbh:right* that of right ones, whose values are *nil* in the initial state.

$$\boxed{e_2}\begin{bmatrix} \mathit{syn}\colon \boxed{r_0}\begin{bmatrix} \mathit{FS2} \end{bmatrix} \\ \mathit{idx}\colon \boxed{m}\, x \\ \mathit{add}\colon \boxed{a_0}\begin{bmatrix} \mathit{left}\colon & \boxed{l} \\ \mathit{right}\colon & \boxed{r} \end{bmatrix} \\ \mathit{nbh}\colon \boxed{n_0}\begin{bmatrix} \mathit{left}\colon & \boxed{ln_2}\,\mathit{nil} \\ \mathit{right}\colon & \boxed{rn_2}\,\mathit{nil} \end{bmatrix} \end{bmatrix} \tag{5.9}$$

The principle governing the structure is typing of channels. The channel m must be independent of others because it is used to receive a channel through which the discourse marker is emitted. Similarly, a_0 and n_0 are of different types. The feature structure can be encoded as a set of processes as is shown in the figure 5.14. The structure of feature descriptions can of course be investigated from a number of points of view. Our approach to the issue is to refer to channel types.

$$\begin{array}{l}
C2 =_{\text{def}} (\nu\, e_1, e_2, e_3)(\\
\quad (\nu\, r_0, m, l, r, ln_2, rn_2, lc_2, rc_2, a_0, n_0, pln_2, prn_2)(\\
\qquad !\, e_2(k, d, f, g_0, h_0).(\ [k = \texttt{syn}]\overline{d}\langle r_0\rangle \\
\qquad\qquad + [k = \texttt{idx}]\overline{f}\langle m\rangle \\
\qquad\qquad + [k = \texttt{add}]\overline{g_0}\langle a_0\rangle \\
\qquad\qquad + [k = \texttt{nbh}]\overline{h_0}\langle n_0\rangle) \\
\qquad \mid\ !\, a_0(k, g_1).([k = \texttt{left}]\overline{g_1}\langle l\rangle \\
\qquad\qquad + [k = \texttt{right}]\overline{g_1}\langle r\rangle) \\
\qquad \mid\ !\, n_0(k, h_1).([k = \texttt{left}]\overline{h_1}\langle ln_2\rangle \\
\qquad\qquad + [k = \texttt{right}]\overline{h_1}\langle rn_2\rangle) \\
\qquad \mid\ FS2(r_0) \\
\qquad \mid\ (\nu\, x)(\ !\, m(y).\overline{y}\langle x\rangle\ ) \\
\qquad \mid\ !\, l(y).C2_l(y) \\
\qquad \mid\ !\, r(z).C2_r(z) \\
\qquad \mid\ !\, ln_2(x).pln_2(y).\overline{pln_2}\langle y\rangle.\overline{x}\langle y\rangle \\
\qquad \mid\ !\, rn_2(x).prn_2(y).\overline{prn_2}\langle y\rangle.\overline{x}\langle y\rangle \\
\qquad \mid\ \overline{pln_2}\langle lc_2\rangle \quad \mid\ !\, Nil(lc_2) \\
\qquad \mid\ \overline{prn_2}\langle rc_2\rangle \quad \mid\ !\, Nil(rc_2)\ ) \\
\quad \mid\ e_1(c_1, l_1, r_1, ln_1, rn_1).\overline{r_1}\langle e_2\rangle \\
\quad \mid\ e_3(c_3, l_3, r_3, ln_3, rn_3).\overline{l_3}\langle e_2\rangle\ )
\end{array}$$

Table 5.14. The definition of $C2$ with the feature *idx*

5.5.3 Integrating multiple information sources

Before concluding the chapter, we show how the proposed formalisation of feature structures as processes is related to our notion of integrating multiple information sources, which has been presented in Introduction. To refresh our memory, Figure 5.17 reproduces Figure 1.9 shown in Introduction. Our goal is to implement the information flow as systems of communicating processes. In the previous chapter 4, we were concerned with semantic part of the construction and have shown how situation types and their anchoring to situations can be described as processes. In this chapter 5, we have worked out the syntax part and have shown how feature structures can be constructed as processes and sentences can be parsed as a result of reactions. The remaining task is to establish the syntax-semantics interface also as processes.

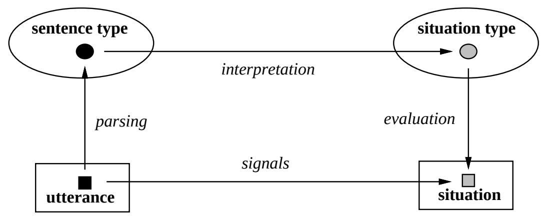


Figure 5.17. The information flow by utterances

Before presenting how we establish the interface, it is useful to ask here why we would like to describe everything as processes. To answer the question, we have to trace back our motivation into Cooper's work on Situation-Theoretic Grammar (STG) [Coo91]. The characteristic point of STG is that it applies the same framework to studying syntax, the framework that is primarily applied to studying semantics. The application becomes possible by regarding utterances as a situation supporting various linguistic information. For example, let u_1 be an utterance of 'man'. The utterance regarded as a situation can support many items of linguistic information, among them are the item that its category is 'noun' and another item that it is characterised phonetically as 'man'. The situation can be expresses in Extended Kamp Notation (EKN) as follows:

(5.10)

u_1
cat(u_1, noun) use-of(u_1, 'man')

STG can be studied in itself, but what most interests us is its connection with Situation Semantics. For the sake of discussion, we consider an example, *"A man walks"*, whose meaning can be expressed as follows:

(5.11)

$r \to X$
man(X) walk(X)

The representation (5.11) is an infon abstracted over the parameter X indexed with the role, r. Presented the representation, one is entitled to ask what the role is. Technically, we can leave the question out by assuming unlimited supply of roles, e.g., using natural numbers as role indices. But more philosophically sound answer is desirable, and one solution proposed by Cooper in [Coo93a] is to regard utterances as role indices. In the above example, we can use the utterance u_1 as the role to index the parameter X as below:

(5.12) a.

$u_1 \to X$
man(X) walk(X)

b.

u_1
cat(u_1, noun) use-of(u_1, 'man')

As Cooper claims, the technical advantage of this approach is that it avoids us having to make some arbitrary decisions about generating enough unique role indices. More importantly, however, there is also a theoretical advantage obtained by combining the items of information concerning syntax and semantics. One application among others presented by Cooper in [Coo93a] is to study how utterances contribute to transferring a meaning from the speaker to the addressee in dialogue. The point is that the utterance situation can be shared between dialogue participants, thus, it serves as a common ground by providing them with common indices to parameters that may be represented differently among participants.

We do not go into the study of dialogue phenomena in the approach. What is insightful for us in the use of utterance situation as a role is that the semantic representation (5.12)a is abstracted over its corresponding syntactic information (5.12)b. We are concerned to know how the relation between syntax and semantics could be formalised. One possibility is to apply the λ-calculus and study the relation in terms of functional

abstraction and application. The other direction, which we present in the following, is to regard the relation as a process.

We complete now our construction of linguistic channel (Figure 5.17) as systems of processes by sketching how the syntax-semantics interface can be described as a process. Adopting our graphic representation, we represent (5.12)a as a system of processes as shown in Figure 5.18. For the sake of simplicity, we adopt here a simple encoding and introduce to our graphic representation some boxes. In the figure, we use r as role and x as parameter. To indicate r and x are bound within the system, we enclose the whole system with the dotted box, putting r and x in upper part. The parts enclosed with the bold box means that these parts are available as many times as requested. As a whole, the figure depicts a system of processes such that x will be replaced by some other individual imported through the role r and the role is accessible through both man' and $walk'$ finitely many times.

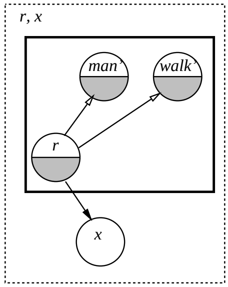


Figure 5.18. A system encoding (5.12)a

To construct the system as the result of interactions, we add to each lexical item encoded as a process a subsystem encoding parts of the meaning. Figure 5.19 depicts the subsystems connected with each other at a certain location. The leftmost part depicts the meaning of 'a' of the sentence, *"A man walks"*. The subsystem contains the part importing some individual through r to replace it for x. The subsystem is abstracted over the role r with an index u, through which it is connected to its right neighbour. The middle part depicts the subsystem encoding the meaning of 'man'. The part to be integrated into the resulting system is abstracted over s with an index v, through which it is connected to its left neighbour. Since we have already explained in this chapter how each subsystems can be connected with each other, we leave the location of interaction anonymous for simplicity. The rightmost part depicts the subsystem encoding the meaning of 'walks'. The part is also abstracted over t with an index w, which is connected to its left neighbour at an anonymous location.

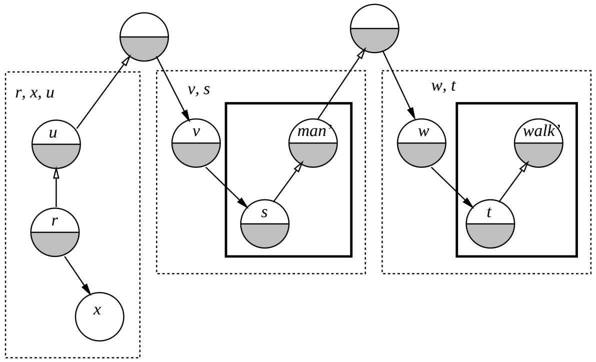


Figure 5.19. The subsystems encoding parts of meaning

We follow briefly the steps from the initial state towards the end result. Firstly, the left two subsystems in the figure interact with each other when the two words, 'a' and 'man', are recognised as forming the noun phrase (Figure 5.20). We assume that the two subsystems are connected in the same way as the subsystems encoding feature structures get into touch with each other. The point here is that the connection between the two subsystems encoding parts of meaning are established as the result of interactions in syntax part. In other words, the way the parts for semantic representations are combined with is determined by the way the parts for syntax is organised. As the result of interaction between the subsystems encoding syntactic information for 'a' and 'man', the two subsystems encoding the parts of meaning is connected with. Then, u replaces v and the scope of bound names are extruded accordingly. Upon the interaction via u, the created subsystem is further evolving into another subsystem encoding the meaning of 'a man' when r replaces s (Figure 5.21).

To continute the interaction, the subsystems encoding the parts of the meaning for 'a man' and 'walks' make a copy of their parts that can be provided with as many as requested (Figure 5.22). The two subsystems interact with each other as the two words are recognised as forming a sentence. Upon the interaction, w is replaced by man', and further t is replaced by r (Figure 5.23). The result is the same as the system shown in Figure 5.18. By removing the man'/w node and conjoining the nodes bearing the same name, r, we obtain the system encoding the meaning of the sentence, *"A man walks"*.

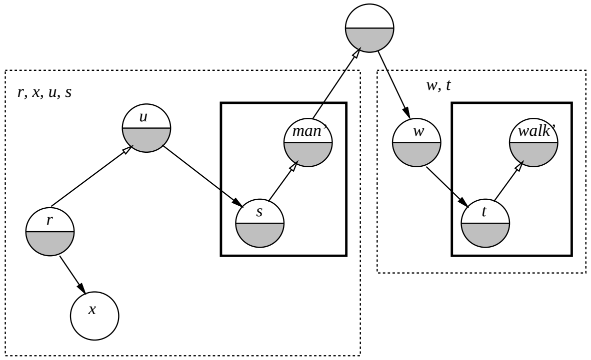


Figure 5.20. The first step of interaction

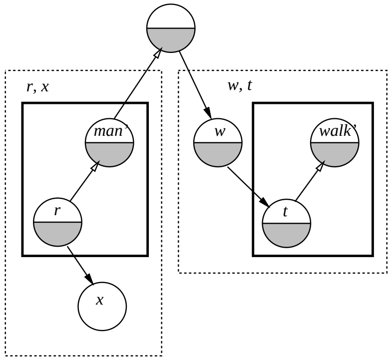


Figure 5.21. The second step of interaction

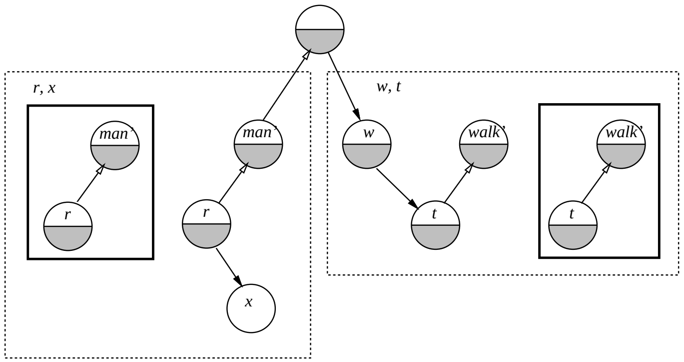


Figure 5.22. The third step of interaction

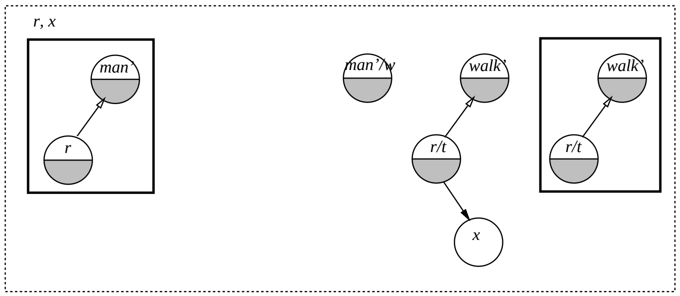


Figure 5.23. The fourth step of interaction

We argue that the above formalisation of the syntax-semantics interface is a refinement of the approach proposed by Cooper in [Coo93a] and that our approach is also a faithful implementation of the relational theory of meaning proposed by Barwise and Perry in [BP83]. Recall that Barwise and Perry argue in the book that the meaning of sentences must be captured in terms of the relation between utterance situations and described situations. The relation plays a central role in Situation Semantics. One can contrast the situation-theory approach with other theories. In Montague-style approach, for example, it is simply assumed that there is some (metaphysical) correspondence between syntax and semantics, i.e., between categories and types. In classical Discourse Representation Theory (DRT) [KR93], the relation is established via a set of rules rewriting syntax trees into Discourse Representation Structures. We can say that in DRT the syntactic information is operationally related to the semantic information, but the ontological status of the rules has never been questioned about. On contrary, the relation between syntax and semantics is part of the meaning in our approach. Recent development in Channel Theory [Bar93, SB93] enables us to speak about the relation in terms of constraints and connections. Our work presented in the dissertation can be understood as implementing the situation/channel theory approach to meaning in process algebra.

We also argue that our approach enables us to describe other factors that may contribute to anchoring semantic representations to described situations. Cooper suggests in [Coo93a] that not only utterance situations but also described situations and resource situations can be used to index parameters. It is also argued there that even a body movement in the utterance situation such as pointing a man could serve as an index. These topics need further research, but we believe that our approach can be applied to modelling all these factors as a process.

We should also mentioned here that representing both syntactic and semantic information in the same framework enables us to study how the way a sentence is parsed can be affected by the preceding semantic representations. Let us consider a short discourse, *"A man walks. He coughs."* How the discourse marker introduced in the first sentence can be exported to the semantic representation for the second sentence? We think that the discourse marker is passed to through the syntactic structure of *"He coughs"*. In other words, the syntactic structure is abstracted over the reference that resides in the preceding semantic representation. We believe that the parser presented in the chapter is expressive enough to encode such a mechanism. For example, the name r used as a role in Figure 5.18 can be freely exported to other processes for parsing upon request provided a suitable port. Detailed study is, however, left for future project.

5.6 Conclusion

In this chapter, we have developed a parser that analyses the syntactic structure of sentences as the result of a course of interactions between agents. We have shown a bottom-up chart parser can be implemented as a system of communicating processes. We have also shown how state transitions of the parser can be inferred using the combinatory intuitionistic linear logic. We have also discussed how semantic representation may be generated and have proposed a way to organise feature structures based on channel types.

The research on concurrent parsing is still under development, and few works have been reported. To investigate and compare proposals to concurrent parsing, we need a theoretical basis which is expressible enough to describe various models. While empirical study is yet to be done, we have shown that the π-calculus can meet the requirement. Not only its fine-grained algebraic theory, but also its relation with linear logic, which we have studied in this thesis, should help us investigate the possibility of concurrency applied to parsing.

Chapter 6

Conclusion

6.1 Summary

In this thesis, we have proposed to regard language use as actions and have refined the information flow by utterances as *linguistic channels*. The information conveyed by an utterance is perfect when the flow is recognised by an agent, who classifies a particular utterance as certain sentence type, maps it to a situation type, then anchors it to a particular situation that the utterance concerns. We have also observed three factors are involved in and may influence the information flow; *discourse* while parsing sentences, *plans and goals* while interpreting sentence types, and *circumstances* while evaluating situation types. To model the interactions between the information flow by utterances and these three factors, we have constructed the linguistic channels as a system of communicating processes.

Our modelling is based on the π-calculus. To study the calculus in logical setting, we turned to combinatorial intuitionistic linear logic and showed how processes defined in the calculus could be specified in the logic. We have constructed semantic objects employed in Situation Semantics in the calculus and have related Situation Semantics with Channel Theory by relying on the translation from processes in the calculus to linear logic. We have then shown how our approach could be applied to a concurrent parser. In the parser, lexical information expressed in feature structures and grammar rules are both encapsulated into a small unit of computation called *agent*, and sentences are analysed through interactions between them. We have also shown how processes corresponding to semantic representations could be created as the result of interactions. Thus, we have seen how the information flow by utterances can be modelled as a system of communicating processes.

6.2 Comparison with other approaches

In this section, we comment on other approaches sharing the same interest with us. As for simultaneous abstractions and Situation Semantics, we share the interest with Peter Ruhrburg, who defines a Simultaneous Abstraction Calculus and formalises it within an axiomatic framework to describe and compare various semantic theories [Ruh96]. As for feature descriptions, constraint-based approaches have been well-known, with which we compare our approach to feature structures. Finally, we relate our approach to Dynamic Semantics.

6.2.1 Simultaneous abstraction calculus

Peter Ruhrburg has recently developed a Simultaneous Abstraction Calculus (SAC) [Ruh96] to overcome the strict ordering requirements of the standard λ-calculus. He has shown using SAC how various semantic theories such as Discourse Representation Theory [KR93] and Situation Theory [BC93] can be defined within an axiomatic, property-theoretic framework [Tur90].

The characteristic point of his approach is that he extends the standard λ-calculus minimally to give a unified picture of various semantic theories where abstractions play an important role. The result is a version of λ-calculus, but objects can be abstracted over more than one parameter simultaneously. For example, to represent an idea, x likes y, the relation holding between x and y, $like(x,y)$, can be abstracted over the two parameters at the same time, i.e., $\lambda\{x,y\}.like(x,y)$, rather than representing it as a unary function such as $\lambda x \lambda y.like(y)(x)$.

SAC is inspired by the way free variables behave in standard logic and chooses to regard them as discourse markers. The core system itself is not so powerful as to encode various semantic theories directly. For example, the alphabetical equivalence does not hold, i.e., $\lambda\{x,y\}.like(x,y) \neq \lambda\{y,x\}.like(y,x)$, but by adding to the core various axioms, one can simulate the behaviours needed to implement semantic theories.

Our starting point is different from SAC in the sense that we seek our model in systems of communicating processes, not in the λ-calculus, although the motivation of looking for more flexible calculi in terms of abstraction and application are shared in common between both approaches. The reason why we turn to the π-calculus is that we are concerned with the ontological status of the role indices, e.g., r_1 and r_2 in $\lambda\{r_1 \to x, r_2 \to y\}.like(x,y)$. In our model, roles are primitive objects, i.e., port or channel, while they are defined in SAC as a special free variable whose behaviour is specified with additional axioms.

There is a trade-off between these two approaches, and they serve well for different purposes. Our model is richer than SAC, but one does not need additional axioms. To use SAC, one has to add to it special axioms, but the core remains to be simple. The former is more efficient than the latter when implemented because additional axioms needed are already compiled in the model. The latter gives us a nice tool to compare different semantic theories. We justify our approach taken here by reminding the reader of our motivation. We are interested in developing the idea to consider utterances as a role into a rigid framework and have established a syntax-semantics interface based on the idea. Comparing various semantic theories is not our intention.

The different choices of formalisms pose another question. We have not enriched the basic axioms and rules of Linear Logic to keep the system as simple as we can, but have proposed a systematic method to translate dataflow graphs into the logic. It may be interesting project to investigate how Ruhrburg's approach can be applied to formalising the computation specified using our dataflow graphs. The formalisation must help us locate our approach among other semantic theories.

6.2.2 Constraint-based approaches to feature descriptions

In constraint-based approaches, linguistic objects are specified with a set of constraints to be satisfied. The advantage is that one only needs to specify partial information he is concerned with and thus it becomes easy to combine various linguistic objects without being bothered with unnecessary details. Here we compare our modelling of feature structures as processes with other constraint-based approaches. In short, constraint-based formalisms are concerned with model theoretic aspects of linguistic objects, while we are interested in proof-theoretic aspects.

As an example of constraint-based approaches, we consider Kasper-Rounds logic (KRL) [KR86] for comparison because it is a pioneering and well-known work in the area. In their approach, feature structures are modelled as directed acyclic graphs (dags), and upon the model, a logic is defined in the same spirit as Hennessey-Milner logic (HML) [HM85]. There is actually a strong connection between KRL and HML because dags can be regarded as a labelled transition system (LTS) and the development of KRL was inspired by HML.[1] It is, therefore, appropriate to discuss the relationship to constraint-based approaches through the comparison with HML. By relating KRL to HML, we can also avoid to going into the detail of KRL, focusing on the essential difference between constraint-based approaches and ours.

The aim of HML/KRL is to specify a semantic object, a LTS/dag, logically. The

[1] Personal communication with Rounds, April 1995.

satisfaction relation between a LTS/dag and a formula is defined logically, e.g., $M \models \phi$, where M is a LTS/dag and ϕ a formula. Besides the advantage of partial specification, the benefit of this approach is that one can check systematically if a particular model satisfies a formula.

While constraint-based approaches are concerned with models of linguistic objects, the issue we would like to investigate by our approach is proof-theoretic, that is, how or whether such objects can be constructed of given parts defined as special axioms. One advantage of adopting intuitionistic linear logics is that it is guaranteed that there exists a model satisfying the formula whenever we can construct it through inference. The property is useful for synthesising correct feature structures automatically. This property should be contrasted with the fact that HML can only be applied for model checking. Our intuitionistic linear logic is, however, not as powerful as HML when it is seen as a specification formalism as we have seen in Chapter 3 (§3.6.5). The problem is open to further study.

It should also be noted that the model-theoretic approach and proof-theoretic approach do not compete with each other, but are complementary. See, for example, that Smolka, one of the major developers of constraint-based approaches [Smo92], turns to the π-calculus to study efficient constraint solving [Smo94a]. The two approaches study different aspects of the same phenomena. The model-theoretic approach is concerned with the specification of problem, while the proof-theoretic approach is concerned with how to solve it.

6.2.3 Dynamic semantics

The most important point advocated by dynamic semantics, e.g., [KR93, GS91b], is to capture the meaning of sentences as the state changes caused by the utterance of sentences. Although approaches to dynamic semantics are divergent, the view to the meaning is shared among them. The difference will only come up by their different definitions and formalisations of states and states changes. In this broader perspective, our approach can be seen as a member of dynamic semantics family. Recall that the state changes can be depicted schematically as:

$$s_0 \xrightarrow{u_1} s_1 \xrightarrow{u_2} s_2 \cdots s_{n-1} \xrightarrow{u_n} s_n$$

where an utterance, u_1, turns the state, s_0, into another, s_1, and each succeeding utterance, u_i, turns s_{i-1} into s_i until u_n turns s_{n-1} into s_n. Each state, s_i, represents a static meaning such as a discourse representation structure. Notice that the scheme

can be seen as a simple process or a labelled transition system. One may see that at the bottom our approach shares the same view as dynamic semantics.

We have, however, enriched the dynamic model further by combining several processes and allowing them to run in parallel. We have also allowed processes to communicate with each other. The notions of concurrency and communications are what we have added to the dynamic view. Introducing parallelism to dynamic semantics framework has been recently considered by Fernando to temper the problem of non-determinism, where he proposes to preserve all alternatives with disjoint union [Fer93]. The disjoint union is rather reminiscent to the additive conjunction in our system, not to the multiplicative conjunction. As we take the multiplicative conjunction to capture parallelism and concurrency, the notion of concurrency is adopted in different senses, and Fernando's notion of concurrency has little to do with our notion of concurrency.

The benefit of enriching the underlying model as we do is that we can study how particular states can be updated by an utterance with the same framework as we apply to studying semantic representations. To show that such an approach is possible to syntax, we have modelled the way an utterance is processed as a state change caused to a system of communicating processes. By formalising the two domains within the same framework, we can study the dynamics of syntax/semantics interface in great detail. Although we have not exploited the possibility in the thesis, we believe that our approach enables one to study the effect of context during semantic construction from syntactic information. The research will be a future project.

6.3 Future work: plans and goals

We have not investigated the influence of plans and goals on linguistic channels because the topic forms another area of study and goes beyond the scope of the thesis. We conclude the thesis by discussing the issue. As we has argued in the first chapter, the plans and goals that an agent has in his mind can influence his utterance. Coming back to the use of the Japanese phrase-final particle, *"tte"*, there is actually a case where the marking is affected by plans. We keep in mind that the term *"repair"* is a category of plan in dialogue.

Figure 6.1 shows an excerpt from the Japanese Map Task Corpus. Here the giver has `mitisirube` (trig point) in his map, while the follower does not. At first, he does not recognize that the follower does not have it in his map and attempts to attract his attention to it (G_1). As he does not have the item, he tells him so (F_1). The point to be noted is that the follower marks the item with a topic marker, `wa`, to contrast his lack of knowledge although he does not know where it is. He should have marked it with the particle, *"tte"*, if the particle was to mark noun phrases whose referent is not known to the speaker. The response from the giver respects the rule, however. Knowing him not having the item in his map, the giver informs him that he has it righthand side of the west lake (G_2, G_3), marking the noun phrase with the particle.

G_1: `dez, mitisirube-no tokoroko-wo`
and at the trig point
F_1: `ez, mitisirube-wa nai-desu`
eh, I don't have a trig point.(`topic`)
G_2: `ez, nai-desu-ka` `ee, jaa, minami-no`
eh, you don't have it. *eh, then, in the west,*
F_2: `hai`
yes
G_3: `numa-no migi-gawa-ni` `mitisirube-tte aru-n-desu-yo`
righthand side of the lake, *there is a trig point.*(`meta`)
F_3: `hai`
yes
G_4: `eh, ue-no, deko-tte-site-iru-tokoro arimasu-yo-ne`
eh, in upper part, you can see a round place, can't you?
F_4: `ah, hai`
ah, yes

Figure 6.1. An example of the use of *"tte"* (2)

The example suggests we need to consider another sort of information than meaning and referent to explain the use of the particle since they are not sufficient to explain the case. The meaning has to do with the schema of shared knowledge updates. That is, we think that the speaker marks a noun phrase when he initiates a repair as for the item. At G_3 in the excerpt, we think the giver marks the noun, `mitisirube` (*trig point*), with the particle to repair their misunderstanding. His next utterance, G_4, indicates that he is still in repair mode, trying to teach him where it is. (Soon later he will give up it and will forget about the item.)

The point may become clearer by looking at the other excerpt (Figure 6.2).[2] The giver here conducts the follower to go leftwards `chotto` (a little). When they first agree to move left (G_1 and F_1), they think the expression is specified enough for their collaboration, thus do not mark it at all. But when they come to move leftwards actually, the giver cannot tell him the precise length (G_2), and the follower initiates his repair by asking about it, marking the expression with the particle (F_2).

[2] The example was referred to in §4.7.2, too, as a ground for parameterising relations.

G_1: `deh, hanshu-kurai sitara` `hidari-ni chotto hidari-ni chotto` `ikun-desu-yo`
and after we go round half way *a little bit towards left (repeat)* *we are going to*
F_1: `chotto chotto hidari-ni`
a little bit towards left
G_2: `hidari-ni` `gomennasai-ne, nan-ka-neh`
towards left *I'm sorry, how can I say...*
F_2: `chotto-tte dono-kurai`
how a little far is it? (`meta`)

Figure 6.2. An example of the use of *"tte"* (3)

Clearly, the use of the particle has to do with their plans and goals for their dialogue; They do not mark the expression until they try to carry out their collaboration to the next step. Note also the particle marks an adverb, `chotto` (*a little*), not a noun phrase. The particle can actually mark other phrases as well such as verb phrases and post-positionals. We can also observe that the marker is usually used only once when the speaker initiates a repair. Once they agree to enter the repair mode, they will no longer mark the expression with the particle. These observations suggest that the meaning of the particle may be better captured in terms of functions in conversation, that is, to inform the other that they will repair their misunderstanding as to the item.

To investigate the influence of plans and goals on language use, we are interested in the approach initiated by Pollack, who considered plans and goals as complex mental attitudes [Pol90]. We may model such complex attitudes as a system of communicating processes. It should not be difficult to implement a planner although we have not investigated how it can be constructed. Take a planner, for instance, that infers transitions of communication states upon utterances. It can be implemented as a production system, which infers state changes by referring to a set of state transition rules.[3] The feature structures encoded in Chapter 5 should be used to represent knowledge.

What can such an modelling in turn benefit the research on planners? Different from logical approach initiated by Cohen and Levesque [CL90], which formalises speech acts theory as a modal logic, we model planners as a reactive system, which reflects actual systems more closely. The difference does not mean that our approach conflicts with logical approach. As we have seen in the thesis, we can conceive of such a logic as a specification language of processes. Once we succeed in relating our model with the logic of actions, we may introduce the notion of imperfect actions to the logic, which must be useful in studying the meaning of actions.

[3] David Traum for example suggested his theory [Tra94] may be implemented in the π-calculus.

We should however note that the notion of plans and goals is used differently in various contexts. Some plans are only concerned with the structure of dialogue, while others with that of tasks. The use of the particle, *"tte"*, may be studied in terms of its function in structuring dialogue plans. There are a lot of things to be investigated in the area of study. We hope we can investigate the issue and can study the interaction between dialogue plans and task plans with our approach.

Appendix A

Molecular languages

A.1 The language $\mathcal{L}_0$

A.1.1 The syntax of $\mathcal{L}_0$

The syntax of $\mathcal{L}_0$ is defined as is shown in the table A.1. The last rule ensures that **1** comes only in consequent of guarding formula. We also omit **1** in molecules because it is always present. This is justified by the rule, $\mathbf{1} : \mathbf{1} \Rightarrow \mathbf{1}$ in the logic. In fact, every execution of a particle can be followed by **1**, which is not explicitly expressed in the language for simplicity.

Some words follow on binding in nested molecules. We ensure a name should be bound by the most inner binding force. For example, in the molecule, $(\nu\, b)[\overline{a}\langle b\rangle, (\nu\, b)[\overline{c}\langle b\rangle]]$, the second particle, $\overline{c}\langle b\rangle$, nested in another molecule is bound by the second inner operator, not by the first. Therefore, it should be compiled to $[\,\overline{a}\langle b_1\rangle, \overline{c}\langle b_2\rangle\,]$.

A.1.2 The semantics of $\mathcal{L}_0$

Actions

The rule for actions is summarised as follows:

- $[\,\alpha, \beta, \ldots, \gamma \restriction C\,]{*}E \searrow_{\alpha} [\,\beta, \ldots, \gamma \restriction C\,]{*}E$

  where α is unguarded with respect to constraints, C, and a substitution environment, E.

A particle, α, is unguarded with respect to constraints, C, and a substitution environment, E, if

$P \longrightarrow$	M	(molecule)
$M \longrightarrow$	$(\nu\, x)[\, N\,]$	(Molecule with restrictions)
	$\mid [\, N\,]$	(Molecule without restrictions)
$N \longrightarrow$	$A \restriction C$	(Molecule with constraints)
	$\mid A$	(Molecule without constraints)
$A \longrightarrow$	α	(particle)
	$\mid P$	(molecule)
	$\mid A, A$	(concurrent)
$\alpha \longrightarrow$	$\overline{x}\langle\vec{y}\rangle$	(exporting)
	$\mid x(\vec{y})$	(importing)
	$\mid \mathbf{1}$	(unit)
$C \longrightarrow$	$\alpha \prec \alpha$	(guarding)
	$\mid [x = y] \prec \alpha$	(match)
	$\mid C, C$	(conjunction)
$\beta \longrightarrow$	α	(particle)
	$\mid \mathbf{1}$	(no particles enabled)

Table A.1. The syntax of $\mathcal{L}_0$

- there is no particle β in the molecule such that $\beta \prec \alpha$ is in C, and
- every match formula in the form of $[x = y] \prec \alpha$, if any, is satisfied with respect to its substitution environment, E.

Interaction

The rule for interaction is summarised as follows, assuming the compiling step:

- $[\, \alpha, \beta, \gamma \restriction C\,] * E \searrow_{\tau} [\, \gamma \restriction C\,] * E'$

  where α and β are a pair of input and output actions connected by a channel, e.g, $\overline{a}\langle b\rangle$ and $a(x)$, unguarded with respect to C and E, and

  E is updated to E' by incorporating the substitution occured upon the interaction, e.g., $\{{}^{b}\!/_{x}\}$. The effect of substitution applies to the molecule, $[\, \gamma \restriction C\,]$.

A.2 Extension towards the full π-calculus

Replication, !, and choice, +, cannot be specified within the logic, but we give corresponding molecular forms for completeness. We also define the language that can express temporal properties, *liveness* and *safety*, which is not part of the π-calculus.

A.2.1 Replication

Syntax is modified by adding the following rule to P:

$$P \longrightarrow\, !\, M \quad \text{(replicable molecule)}$$

Semantics is changed for interaction. We add to the rule the following rules, which replicates a molecular if there is an interactable molecular with it.

- $[\, !\,[\alpha], \beta, \gamma \restriction C\,] * E \equiv [\, !\,[\alpha], [\alpha], \beta, \gamma \restriction C\,] * E$

  where both α and β are ready to interact with each other, unguarded with respect to C and E

A.2.2 Choice

Syntax is modified by adding the following rule to P:

$$P \longrightarrow M + M \quad \text{(non-deterministic molecule)}$$

Semantics is changed for action by adding to it the following rule. The change to the rule for interaction is trivial.

- $[\, \alpha + \beta, \ldots, \gamma \restriction C\,]{*}E \searrow_{\alpha} [\, \ldots, \gamma \restriction C\,]{*}E$

  where either α or β is executed, unguarded with respect to constraints, C, and a substitution environment, E.

A.2.3 Temporal property

Syntax is modified by adding to C the following rules:

$$\begin{array}{lll} C \longrightarrow & \alpha = \beta & \text{(liveness)} \\ & \mid \alpha \neq \beta & \text{(safety)} \end{array}$$

Semantics is changed for interaction. Only the interactions satisfying temporal property are allowed to occur.

- $[\, \alpha, \beta, \gamma \restriction C\,] * E \searrow_{\tau} [\, \gamma \restriction C\,] * E'$

  where α and β are a pair of input and output actions connected by a channel, e.g, $\overline{a}\langle b\rangle$ and $a(x)$, unguarded with respect to C and E,

  the effect on substitution environments upon the interaction does not violate C,

  E is updated to E' by incorporating the substitution occured upon the interaction, e.g., $\{{}^{b}\!/_{x}\}$. The effect of substitution applies to the molecule, $[\, \gamma \restriction C\,]$.

Bibliography

[ABB+91] Anne H. Anderson, Miles Bader, Ellen G. Bard, Elizabeth H. Boyle, Gwyneth M. Doherty, Simon C. Garrod, Stephen D. Isard, Jacqueline C. Kowtko, Jan M. McAllister, Jim Miller, Catherine F. Sotillo, Henry S. Thompson, and Regina Weinert. The HCRC Map Task Corpus. *Language and Speech*, 34(4):351–366, 1991.

[Abr91] Samson Abramsky. Domain theory in logical form. *Annals of Pure and Applied Logic*, 51:1–77, 1991.

[Acz88] Peter Aczel. *Non-Well-Founded Sets*. Center for the Study of Language and Informaiton, Stanford, California, 1988.

[Agh86] Gul A. Agha. *Actors : a model of concurrent computation in distributed systems*. The MIT Press series in artificial intelligence. MIT Press, Cambridge, Mass., 1986.

[Ahn94] René Ahn. Dynamic knowledge states in type theory. In Bunt et al. [BMR94], pages 1–10.

[AIK+94] Motoko Aono, Akira Ichikawa, Hanae Koiso, Shinji Sato, Makiko Naka, Syun Tutiya, Kenji Yagi, Naoya Watanabe, Masato Ishizaki, Michio Okada, Hiroyuki Suzuki, Yukiko Nakano, and Keiko Nonaka. The Japanese Map Task Corpus: an interim report. In *Spoken Language Processing*, volume SLP3(5), pages 25–30. Information Processing Society of Japan, 1994. in Japanese.

[AIKP93] Peter Aczel, David Israel, Yasuhiro Katagiri, and Stanly Peters, editors. *Situation Theory and its Applications*, volume 3. Center for the Study of Language and Informaiton, Stanford, California, 1993.

[AL91] Peter Aczel and Rachel Lunnon. Universes and parameters. In Jon Barwise, Jean Mark Gawron, Gordon Plotkin, and Syun Tutiya, editors, *Situation Theory and its Applications*, volume 2, pages 3–24. Center for the Study of Language and Informaiton, Stanford, California, 1991.

[Aus62] John L. Austin. *How to do things with words*. Oxford University Press, Oxford, 1962.

[AV93] Samson Abramsky and Steven Vickers. Quantales, observational logic and process semantics. *Mathematical Structures in Computer Science*, 3:161–227, 1993.

[Bar89] Jon Barwise. On the model theory of common knowledge. In *The Situation in Logic*, pages 201–220. Center for the Study of Language and Informaiton, 1989.

[Bar92] Jon Barwise. Information links in domain theory. In S. Brookes, M. Main, A. Melton, M. Mislove, and D. Schmidt, editors, *Proceedings of the Mathematical Foundations of Programming Semantics Conference*, volume 598 of *LNCS*, pages 168–192. Springer Verlag, 1992.

[Bar93] Jon Barwise. Constraints, channels, and the flow of information. In Aczel et al. [AIKP93], pages 3–27.

[BB92] Gérard Berry and Gérard Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96:217–248, 1992.

[BC91] Jon Barwise and Robin Cooper. Simple situation theory and its graphical representation. In Jerry Seligman, editor, *Partial and Dynamic Semantics III*, pages 38–74. Centre for Cognitive Science, University of Edinburgh, 1991. DYANA Report R2.1.C.

[BC93] Jon Barwise and Robin Cooper. Extended kamp notation: a graphical notation for situation theory. In Aczel et al. [AIKP93], pages 29–53.

[BC94] Gérard Boudol and Ilaria Castellani. Flow models of distributed computations: three equivalent semantics for CCS. *Information and computation*, 114:247–314, 1994.

[BE87] Jon Barwise and John Etchemendy. *The Liar: An Essay on Truth and Circularity*. Oxford University Press, 1987.

[BE90] Jon Barwise and John Etchemendy. Information, infons and inference. In Robin Cooper, Kuniaki Mukai, and John Perry, editors, *Situation Theory and its Applications*, volume 1, pages 33–78. Center for the Study of Language and Informaiton, Stanford, California, 1990.

[BHS94] Norbert Bröeker, Udo Hahn, and Susanne Schacht. Concurrent lexicalized dependency parsing: The *ParseTalk* model. In *15th International Conference on Computational Linguistics*, volume 2, pages 379–385. 1994.

[Bie95] G. M. Bierman. What is a categorical model of intuitionistic linear logic? In *Proceedings of International Conference on Typed Lambda Calculi and Applications*, Edinburgh, Scotland, April 1995. to appear in Springer Lecture Notes in Computer Science.

[Bla92] Alan W. Black. *A situation theoretic approach to computational semantics*. PhD thesis, Department of Artificial Intelligence, University of Edinburgh, 1992.

[BMM+94] J. Bos, E. Mastenbroek, S. McGlashan, S. Millies, and M. Pinkal. A compositional DRS-based formalism for NLP applications. In Bunt et al. [BMR94], pages 21–30.

[BMR94] Harry Bunt, Reinhard Muskens, and Gerrit Rentier, editors. *International Workshop on Computational Semantics*, Institute for Language Technology and Artificial Intelligence, Tilburg University, The Netherlands, December 1994.

[BP83] Jon Barwise and John Perry. *Situations and Attitudes*. MIT Press, Cambridge, Mass., 1983.

[BS92] Jon Barwise and Jerry Seligman. The rights and wrongs of natural regularity. In James Tomberlin, editor, *Philosophical Perspectives*. 1992. in preses.

[BSSH94] Norbert Bröeker, Michael Strube, Susanne Schacht, and Udo Hahn. Coarse-grained parallelism in natural language understanding: parsing as message passing. In *The International Conference on New Methods in Language Processing (NeMLaP)*, volume 2, pages 182–189. September 1994. Manchester, U.K.

[Car92] Jean Carletta. *Risk-taking and Recovery in Task-Oriented Dialogue*. PhD thesis, Department of Artificial Intelligence, University of Edinburgh, 1992.

[Cho65] Noam Chomsky. *Aspects of the theory of syntax*. MIT Press, Cambridge, Mass., 1965.

[CL90] Philip R. Cohen and Hector J. Levesque. Rational interaction as the basis for communication. In Philip R. Cohen, Jerry Morgan, and Martha E. Pollack, editors, *Intentions in communication*, pages 221–255. MIT Press, Cambridge, MA, 1990.

[Cla92] Herbert H. Clark. *Arenas of language use*. The University of Chicago Press and Center for the Study of Language and Information, Chicago, 1992.

[CM92] Herbert H. Clark and Catherine R. Marshall. Definite reference and mutual knowledge. [Cla92], pages 9 – 59.

[Coo91] Robin Cooper. Three lectures on situation theoretic grammar. In Miguel Filgueiras, Luis Damas, Nelma Moreira, and Ana Paula Tomás, editors, *Proceedings of Natural Language Processing, EAIA 90*, number 476 in Lecture Notes in Artificial Intelligence, pages 101–140. Springer Verlag, 1991.

[Coo92] Robin Cooper. A working person's guide to situation theory. In Steffen Leo Hansen and Finn Soerensen, editors, *Topics in Semantic Interpretation*, Samfundslitteratur, Frederiksberg, Denmark, 1992.

[Coo93a] Robin Cooper. Integrating different information sources in linguistic interpretation. In *First International Conference on Linguistics at Chosun University*, pages 79–109, Kwangju, Korea, November 1993. Foreign Culture Research Institute, Chosun University.

[Coo93b] Robin Cooper. Towards a general semantic framework. In Robin Cooper, editor, *Integrating Semantic Theories*. ILLC/Department of Philosophy, University of Amsterdam, 1993. Deliverable R2.1.A, Dyana-2.

[Dam93] Mads Dam. Model checking mobile processes. In *Proceedings of 4th International Conference on Concurrency Theory*, volume 715 of *LNCS*, pages 22 – 36. Springer Verlag, August 1993.

[Dam94] Mads Dam. Process-algebraic interpretations of positive linear and relevant logics. *Journal of Logic and Computation*, 4(6):939–973, 1994.

[EW90] Uffe Engberg and Glynn Winskel. Petri nets as models of linear logic. volume 431 of *LNCS*, pages 147–161, Copenhagen, Denmark, May 1990. Springer Verlag.

[EW93] Uffe Engberg and Glynn Winskel. Completeness results for linear logic on petri nets. MFCS'93, Gdańsk, Poland, August 30 - September 3, 1993.

[Fer93] Tim Fernando. Generalized quantifiers as second-order programs — "dynamically" speaking, naturally. In *Proceedings of the Amsterdam Colloquium*. 1993.

[Fre93] Gottlob Frege. Über sinn und bedeutung. In *Zeitschrift für Philosophie und philosophische Kritik*, volume 100, pages 25–50. 1893.

[Fuj94] Tsutomu Fujinami. Representations as processes: situation-theoretic objects encoded in the π-calculus. In Bunt et al. [BMR94], pages 91–100.

[Fuj96a] Tsutomu Fujinami. A dynamic syntax-semantics interface. In *Proceedings of the second Conference on Information-Theoretic Approaches to Logic, Language, and Computation*, pages 63–72. London Guildhall University, July 1996.

[Fuj96b] Tsutomu Fujinami. A process algebraic approach to situation semantics. In Paul Dekker and martin Stokhof, editors, *Proceedings of the 10th Amsterdam Colloquium*, volume 2, pages 263–282, The Netherlands, December 1996. ILLC/Department of Philosophy, University of Amsterdam. ftp://ftp.ims.uni-stuttgart.de/pub/papers/tsutomu.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–102, 1987.

[Gir95] Jean-Yves Girard. Linear logic: its syntax and semantics. In Jean-Yves Girard, Yves Lafont, and Laurent Regnier, editors, *Advances in Linear Logic*, pages 1–42. Cambridge University Press, 1995.

[Gla90] Sheila R. Glasbey. Tense and aspect in natural language processing - a situation theory approach. Master's thesis, Department of Artificial Intelligence, University of Edinburgh, 1990.

[GM84] Ursula Goltz and Alan Mycroft. On the relationship of CCS and petri nets. volume 172 of *LNCS*, pages 196–208, Antwerp, Belgium, July 1984. Springer Verlag.

[GS91a] Jeroen Groenendijk and Martin Stokhof. Dynamic Montague grammar. In Martin Stokhof, Jeroen Groenendijk, and David Beaver, editors, *Quantification and Anaphora I*. Centre for Cognitive Science, University of Edinburgh, 1991. DYANA Report R2.2.A.

[GS91b] Jeroen Groenendijk and Martin Stokhof. Dynamic predicate logic. *Linguistic and Philosophy*, 14(1):39–100, 1991.

[HM85] Matthew Hennessy and Robin Milner. Algebraic laws for non-determinism and concurrency. *Journal of the Association for Computer Machinery*, 32(1):137–161, January 1985.

[Kam84] Hans Kamp. A theory of truth and semantic representation. In J. A. G. Groenendijk, T. M. V. Janseen, and M. Stokhof, editors, *Truth, Interpretation and Information: Selected Papers from the Third Amsterdam Colloquium*, pages 1–41. Dordrecht: Foris Publication, 1984.

[KR86] Robert T Kasper and William C Rounds. A logical semantics for feature structures. In *Proceedings of 24th ACL*, pages 257–266, 1986.

[KR93] Hans Kamp and Uwe Reyle. *From Discourse to Logic: Introduction to Model theoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory*. Dordrecht: Kluwer, 1993.

[Kun87] Susumu Kuno. *Functional Syntax*. University of Chicago Press, Chicago, 1987.

[Laf88] Yves Lafont. The linear abstract machine. *Theoretical Computer Science*, 59:157–180, 1988.

[Lar90] Kim Guldstrand Larsen. Ideal specification formalism = expressivity + compositionality + decidability + testability + $\cdots$. volume 458 of *LNCS*, pages 33–56. Springer Verlag, 1990.

[Mil83] Robin Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.

[Mil89] Robin Milner. *Communication and Concurrency*. Prentice Hall, New York, 1989.

[Mil92a] Dale Miller. The π-calculus as a theory in linear logic: Preliminary results. In E. Lamma and P. Mello, editors, *Proceedings of Workshop on Extensions to Logic Programming*, volume 660 of *LNCS*, pages 245–265. Springer Verlag, October 1992.

[Mil92b] Robin Milner. Action structures. Technical Report ECS-LFCS-92-249, Department of Computer Science, University of Edinburgh, December 1992.

[Mil92c] Robin Milner. Functions as processes. *Journal of Mathematical Structures in Computer Science*, 2(2):119–141, 1992.

[Mil93a] Robin Milner. Action structures and the π-calculus. Technical Report ECS-LFCS-93-264, Department of Computer Science, University of Edinburgh, May 1993.

[Mil93b] Robin Milner. The polyadic π-calculus: a tutorial. In F. L. Bauer, W. Brauer, and H. Schwichtenberg, editors, *Logic and Algebra of Specification*, pages 203–246. Springer Verlag, 1993.

[Mil94] Robin Milner. Pi-nets: a graphical form of π-calculus. In *Proceedings of ESOP '94*, volume 788 of *LNCS*, pages 26–42. Springer Verlag, 1994.

[Mil95] Robin Milner. Calculi for interaction. Unpublished ms., April 1995.

[MOM91] Narcìso Martí-Oliet and José Meseguer. From petri nets to linear logic. *Mathematical Structures in Computer Science*, 1:69–101, 1991.

[MPW91] Robin Milner, Joachim Parrow, and David Walker. Modal logics for mobile processes. In Jos C. M. Beaten and Jan Frisco Groote, editors, *Proceedings of 2nd International Conference on Concurrency Theory*, volume 527 of *LNCS*, pages 45 – 60. Springer Verlag, August 1991.

[MPW92] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, parts I and II. *Information and Computation*, 100:1–40 and 41–77, 1992.

[MS94] L. Moss and Jerry Seligman. Classification domains and information links: a brief survey. In J van Eijck and A Visser, editors, *Logic and Information Flow*. MIT Press, 1994.

[Mus92] Reinhard Muskens. Anaphora and the logic of change. Technical Report 34, Institute for Language Technology and Artificial Intelligence, Tilburg University, The Netherlands, 1992.

[Mus93] Reinhard Muskens. A compositional discourse representaiton theory. In *Proceedings of the Amsterdam Colloquium*, pages 467–486. 1993.

[Mus94] Reinhard Muskens. Categorial grammar and discourse representation theory. In *15th International Conference on Computational Linguistics*, volume 1, pages 508–514. 1994.

[NPS91] Hideyuki Nakashima, Stanley Peters, and H Schütze. Communication and inference through situations. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, volume 1, pages 75–91. 1991.

[Oka93] Mitsuhiro Okada. Mobile linear logic as a framework for asynchronous and synchronous mobile communication calculi (preliminary report). Unpublished ms., March 1993.

[Pie94] B. C. Pierce. Programming in the pi-calculus: An experiment in concurrent language design. Technical report, Computer Science Department, University of Edinburgh, 1994. Tutorial Notes for PICT.

[Plo81] Goldon D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI-FN-19, Computer Science Department, Aarhus University, 1981.

[Pol90] Martha E. Pollack. Plans as complex mental attitudes. In Philip R. Cohen, Jerry Morgan, and Martha E. Pollack, editors, *Intentions in communication*, pages 77–103. MIT Press, Cambridge, MA, 1990.

[Pra81] V. R. Pratt. Dynamic logic. In J. W. de Bakker and J. van Leeuwen, editors, *Foundations of Computer Science III Part II*, volume 109 of Mathematical Centre Tracts, pages 53–84. 1981. Centrum voor Wiskunde en Informatica, Amsterdam.

[PS94] Carl Pollard and Ivan A. Sag. *Head-Driven Phrase Structure Grammar*. The University of Chicago Press and Center for the Study of Language and Information, Chicago, Ill. and Stanford, Ca., 1994.

[Ran91] Aarne Ranta. Intuitionistic categorial grammar. *Linguistics and Philosophy*, 14:203–239, 1991.

[Ruh96] Peter Ruhrberg. *Simultaneous Abstraction and Semantic Theories*. PhD thesis, Centre for Cognitive Science, University of Edinburgh, Edinburgh, 1996.

[SB93] Jerry Seligman and Jon Barwise. Channel theory: toward a mathematics of imperfect information flow. Unpublished ms., May 1993.

[Sch92] Emanuel A. Schegloff. Repair after next turn: The last structurally provided defense of intersubjectivity in conversation. *American Journal of Sociology*, 97(5):1295–1345, 1992.

[SHB94] Susanne Schacht, Udo Hahn, and Norbert Bröeker. Concurrent lexicalized dependency parsing: A behavioral view on *ParseTalk* events. In *15th International Conference on Computational Linguistics*, volume 2, pages 489–493. 1994.

[Smo92] Gert Smolka. Feature constraint logics for unification grammars. *Journal of Logic Programming*, 12:51–87, 1992.

[Smo94a] Gert Smolka. A foundation for concurrent constraint programming. In *Constraints in Computational Logics*, volume 845 of *LNCS*, pages 50–72. Springer Verlag, September 1994.

[Smo94b] Gert Smolka. An Oz primer. Technical report, German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany, 1994. available from http://ps-www.dfki.uni-sb.de/oz/.

[Sti92] Colin Stirling. Modal and temporal logics for processes. Technical Report ECS-LFCS-92-221, Department of Computer Science, University of Edinburgh, June 1992.

[Tak89] Yukinori Takubo. Meishikku-no modality (the modality of noun phrases). In *Nihongo-no modality (The modality in Japanese)*, pages 211–233. Kurosio, Tokyo, 1989. in Japanese.

[Tak90] Yukinori Takubo. On the role of hearer's territory of information: A contrastive study of dialogic structure in Japanese, Chinese, and English as manifested in the third person pronoun system. In *Advances in Japanese Cognitive Science*, volume 3, pages 67–84. Kodansha Scientific, Tokyo, 1990. in Japanese.

[TK92] Yukinori Takubo and Satoshi Kinsui. Discourse management in terms of mental domains. Unpublished ms., to appear in *Travaux de Linguistique Japonaise*, 1992.

[Tra94] David R. Traum. *A computational theory of grounding in natural language conversation.* PhD thesis, Computer Science Department, University of Rochester, New York, 1994.

[Tro92] Anne Sjerp Troelstra. *Lectures on linear logic*, volume 29. Center for the Study of Language and Informaiton, Stanford, California, 1992.

[Tur90] Raymond Turner. *Truth and Modality for Knowledge Representation.* Pitman, London, 1990.

[Wal95] David Walker. Objects in the π-calculs. *Information and Computation*, 116(2):253–271, 1995.