On resources and tasks

Giorgi Japaridze

Department of Computing Sciences, Villanova University,
800 Lancaster Avenue, Villanova, PA 19085, USA

Abstract

Essentially being an extended abstract of the author's 1998 PhD thesis, this paper introduces an extension of the language of linear logic with a semantics which treats sentences as *tasks* rather than true/false statements. A resource is understood as an agent capable of accomplishing the task expressed by such a sentence. It is argued that the corresponding logic can be used as a planning logic, whose advantage over the traditional comprehensive planning logics is that it avoids the representational frame problem and significantly alleviates the inferential frame problem.

Dedicated to the dear memory of **Leri Mchedlishvili** (1936-2013): my first teacher of logic, colleague, and a prominent figure in Georgian science and education.

1 Introduction

This paper is being revived after having been frozen, in an unfinished form, for more than a decade. It is essentially an extended abstract of the author's 1998 dissertation [10]. For "historical" considerations, its original style and language are almost fully preserved.

Since the birth of Girard's [5] linear logic, the topic of substructural logics, often called "resource logics", has drawn the attention of many researchers, with various motivations and different traditions.[1] The common feature of these logics is that they are sensitive with respect to the number of occurrences of subformulas in a formula or a sequent, the most demonstrative example of which is the failure of the classical principles

$$A \to A\&A$$

and

$$\frac{A \to B \qquad A \to C}{A \to B\&C}$$

as a result of removing the rule of contraction from classical sequent calculus.

The philosophy behind this approach is that if formulas are viewed as resources, the conjunction is viewed as an operator which "adds up" resources and the implication is viewed as an operator which converts one resource (the antecedent) into another (the consequent), then $A\&A$ is generally stronger than A, and $A \to (B\&C)$ is stronger than $(A \to B)\&(A \to C)$. For example, \$1&\$1 should be understood as being equivalent to \$2 rather than \$1, so that $\$1 \to (\$1\&\$1)$ is not valid; one cannot get both a can of Coke and a sandwich for \$1 even if each of them costs a dollar, so

$$\frac{\$1 \to coke \qquad \$1 \to sandwich}{\$1 \to (coke \ \& \ sandwich)}$$

[1] An extensive survey of substructural logics can be found in [4].

fails, too.

Although this kind of resource philosophy seems intuitively very clear, natural and appealing, it has never been fully formalized, and substructural logics owe their name "resource logics" mostly to certain syntactic features rather than some strict and, at the same time, intuitively convincing resource semantics behind them. The present work is an attempt to develop such a semantics.

1.1 Resources, informally

The simplest type of resources, which we call *unconditional resources*, consist of 2 components: *effect* and *potential*. Effect is the process associated with and supported by the resource, and potential is the set of resources into which the resource can be converted at the owner's wish. The effect of the resource "My car in my driveway" is the process "my car is in my driveway", and its potential, at a high level of abstraction, is {"My car on the Ross bridge", "My car on the Franklin Bridge", "My car at the airport",...}. For some resources, such as $2 \times 2 = 4$, the potential can be empty. And some resources can be "effectless" — their value is associated only with the resources into which they can be converted. Money can be viewed as an example of such a resource: its value is associated only with its convertibility into "real" things.

The elements of the potential can be viewed as the *commands* that the resource accepts from its owner. My computer, when it is shut down, maintains the process "the screen is dark" as its effect. In this state it accepts only one command: "start". After this command is given, it turns into another resource, — let us say "another resource" rather than "another state", — whose effect is "the initial menu is on the screen" and whose potential consists of the commands "Word Processor", "Games", "Netscape", "Telnet",... When I give one of these commands, it turns yet into a new resource, etc.

It might be helpful to think of resources as *agents* carrying out certain *tasks* for us. Mathematically, task is a synonym of resource, and which of these two words we use, depends on the context. Between agents a master-slave (ownership) relationship can hold. What is a task for the slave, is a resource for the master.

Thus, intuitively, an unconditional resource is an agent which maintains a certain process (its effect) and also accepts commands (elements of its potential) from its master and executes them, where executing a command means turning into a certain new resource.

1.2 Resource conjunction

Let us consider some more precise examples. Let Φ be an agent which writes in memory location $L1$ any number we tell it to write, and then keeps this number there until we give it a new command of the same type again, and so on. Initially, this resource maintains the value 0 in $L1$.

This is an example of an inexhaustible resource — it can execute our commands as many times as we like.

Consider another agent Ψ which writes in location $L2$, when we give it such a command, the factorial of the current value of $L1$, and keeps that number there (even if the value of $L1$ changes meanwhile) until we give it a new command of the same type. Unlike Φ, this resource accepts only one command, even though, again, infinitely many times. Initially it keeps the value 0 in $L2$.

We denote the conjunction operator for resources by $\ddot{\wedge}$. What would $\Psi \ddot{\wedge} \Phi$ mean? Intuitively, having the conjunction of two agents means having them both as independent resources, so that we can use each of them as we wish, without affecting our ability to use the other. Initially this resource maintains the value 0 in locations $L1$ and $L2$. In every state, it accepts two types of commands: 1) Write in $L2$ and maintain there the factorial of the current value of $L1$ (only one command), and 2) Write in $L1$ and maintain there number n (one command per number).

1.3 Conditional resources

Both Ψ and Φ, as well as their conjunction $\Psi \ddot{\wedge} \Phi$, are examples of unconditional resources: they maintain their effects and execute commands unconditionally, without asking anything in return. However, in real life, most resources are *conditional.* My car can travel, but now and then it will require from me to fill up its tank; my computer will copy any file to a floppy disk, but only if I execute its countercommand "Insert a floppy disk into drive A".

We use the symbol $\ddot{\rightarrow}$ to build expressions for conditional resources. Having the resource $\Theta_1 \ddot{\rightarrow} \Theta_2$ means that I can make it work as Θ_2 if I can give to it the resource Θ_1. It is not necessary to actually assign some agent accomplishing Θ_1 to it. I can assign to it a "virtual Θ_1", which means that all I need in order to make this conditional resource work as Θ_2 is to execute every command it issues for Θ_1. So, $\Theta_1 \ddot{\rightarrow} \Theta_2$ can be seen as a resource which consumes the resource Θ_1 and produces the resource Θ_2, or converts the resource Θ_1 into Θ_2.

Consider one more unconditional resource, Γ, which writes in memory location $L2$ the factorial of any number we give to it, and maintains it there until it gets a new command of the same type. Just like Ψ, initially it maintains 0 in $L2$.

Can I accomplish Γ as a task? Generally — not. Even if I can compute factorials in my head, I may not have writing access to location $L2$ after all, or I may have this access but some other agent can have that kind of access, too, and can overwrite the number I needed to maintain in $L2$.

However, if the resources Φ and Ψ are at my disposal, then I can carry out Γ. Whatever number my master gives me, I first make Φ write it in $L1$, and then make Ψ write and maintain its factorial in $L2$. So, I cannot accomplish the task Γ, but I can accomplish the task

$$\Phi \ddot{\wedge} \Psi \ \ddot{\rightarrow} \ \Gamma,$$

which is an example of a conditional resource.

If we go to lower levels of abstraction, it may turn out that, say, Ψ, itself, is (the consequent of) a conditional resource. It may require some memory space, ability to write and read and perform some arithmetic operations there, etc. Let us denote this resource, — the resource required by Ψ to function successfully, — by Δ. In that case, the resource I possess is

$$\Phi \ \ddot{\wedge} \ (\Delta \ddot{\rightarrow} \Psi)$$

rather than $\Phi \ \ddot{\wedge} \ \Psi$. I have no reason to assume that I can carry out Γ now. However, I can carry out

$$\Big(\Delta \ \ddot{\wedge} \ \Phi \ \ddot{\wedge} \ (\Delta \ddot{\rightarrow} \Psi)\Big) \ \ddot{\rightarrow} \ \Gamma.$$

Because I can use the conjunct Δ to do whatever $(\Delta \ddot{\rightarrow} \Psi)$ wants from its Δ, and thus make that conditional resource work as Ψ.

What if Φ, too, requires Δ as a resource? That is, can I successfully handle the task

$$\Big(\Delta \ \ddot{\wedge} \ (\Delta \ddot{\rightarrow} \Phi) \ \ddot{\wedge} (\Delta \ddot{\rightarrow} \Psi)\Big) \ \ddot{\rightarrow} \ \Gamma?$$

No, even though, by classical logic, the above formula follows from

$$\Big(\Delta \ \ddot{\wedge} \ \Phi \ \ddot{\wedge} \ (\Delta \ddot{\rightarrow} \Psi)\Big) \ \ddot{\rightarrow} \ \Gamma.$$

I cannot, because I have only one Δ while I need two. What if the two conditional agents $\Delta \ddot{\rightarrow} \Phi$ and $\Delta \ddot{\rightarrow} \Psi$ issue conflicting commands for Δ? For example, the first one may require to write in a certain location $L3$ the number 13 and maintain it there, while the other needs 14 to be maintained in that location? One location cannot keep two different values. In other words, Δ could serve one master, but it may not be able to please two bosses simultaneously.

And not only because conflicting commands may occur. Certain resources can execute certain commands only once or a limited number of times. A kamikaze can attack any target, and the commands "attack A" and "attack B" are not logically conflicting; however, I cannot carry out the task of 2 kamikazes if I only have 1 kamikaze at my command: after making him attack A, he will be gone.

This is where linear-logic-like effects start to kick in. As we have just observed, the principle $\Theta \ddot{\rightarrow} \Theta \ddot{\wedge} \Theta$ is not valid. On the other hand, all the principles of linear logic + weakening are valid.

1.4 More on our language

In addition to $\ddot{\rightarrow}$ and $\ddot{\wedge}$, we need many other connectives to make our language sufficiently expressive. When we described Φ and Ψ, we just used English. But our formal language should be able to express all that in formulas. In fact the language is going to be much more expressive than the language of linear logic.

The formulas of our language are divided into 3 categories/levels: facts, processes, and resources. Each level has its own operators and may use lower-level expressions as subexpressions.

Facts are nothing but classical first order formulas, with their usual semantics. We use the standard Boolean connectives and quantifiers (without dots over them) to build complex facts from simpler ones.

We assume that *time* is a linear order of *moments*, with a beginning but no end. An *interval* is given by a pair (i, j), where i is a time moment and j is either a greater time moment or ∞.

While facts are true or false at time moments, *processes* are true or false on intervals.

The Boolean connectives and quantifiers are applicable to processes, too. To indicate that they are process operators, we place one dot over them. $\alpha \dot{\wedge} \beta$, where α and β are processes, is the process which is true on an interval if and only if both α and β are true on that interval. The meaning of the other "dotted" connectives ($\dot{\rightarrow}$, $\dot{\neg}$, $\dot{\forall}$, ...) should also be clear. They behave just like classical connectives, — all the classical tautologies, with dots over the operators, hold for processes, too. But, as we have already noted, this is not the case for resources.

Here are some other process operators:

$\Uparrow A$, where A is a fact, is a process which holds on an interval iff A is true at every moment of the interval except, perhaps, its first moment.

$\angle A$, where A is a fact, is true on an interval iff A is true at the first moment of the interval.

$\alpha \rhd \beta$, where α and β are resources, is true on an interval iff α holds on some initial segment of the interval and β holds on the rest of the interval; in other words, if the process α switches to the process β at some internal moment of the interval.

As for the resource level expressions, they, too, use classical operators, with a double dot over them. We have already seen the intuitive meaning of two of them, $\ddot{\wedge}$ and $\ddot{\rightarrow}$. The other basic resource-building operator is $\gg$. The expression

$$\alpha \gg (\Delta_1, ..., \Delta_n),$$

where α is a process and the Δ_i are resources, stands for the resource whose effect is α and whose potential is $\{\Delta_1, ..., \Delta_n\}$. The expression

$$\alpha \gg x\Delta(x)$$

is used to express resources with possibly infinite potentials: the potential of this resource is $\{\Delta(a) : \ a \in D\}$, where D is the domain over which the variable x ranges.

To be able to express infinite resources such as Φ and Ψ (Figures 1 and 2), we also need to allow recursively defined expressions. Let

$$\Phi' := \ \gg x\Big(\big(\Uparrow L1(x)\big)\Phi'\Big)$$

and

$$\Psi' := \ \gg \Big(\big(\dot{\exists} x(\angle(L1(x)\dot{\wedge} \Uparrow L2(!x)\big)\Psi'\Big).$$

Then, resource Φ can be expressed by $\big(\Uparrow L1(0)\big)\Phi'$ and resource Ψ can be expressed by $\big(\Uparrow L2(0)\big)\Psi'$.

For readers familiar with linear logic, we will note that $\gg$ is in fact a generalization of the additive conjunction or quantifier (while $\ddot{\wedge}$ and $\ddot{\rightarrow}$ correspond to the multiplicative conjunction and implication). The generalization consists in adding one more parameter, α, to this sort of conjunction. The standard linear-logic additive conjunction should be viewed as a special case of $\gg$-formulas where the left argument of $\gg$ is a trivial process, such as $\Uparrow \top$.

The semantics of $\gg$ is that it is a "manageable $\rhd$". If in $\alpha \rhd \beta$ the transfer from α to β happens "by itself" at an arbitrary moment, in the case of $\alpha \gg (\Psi_1, ..., \Psi_n)$ the transfer from α to the effect of Ψ_i happens according to our command. But at what moment should this transfer occur? If we assume that exactly at the moment of giving the command, then even the principle $\Delta \ddot{\rightarrow} \Delta$ can fail, because execution of a command, or passing the command which I receive in the right Δ to the left Δ always takes time. Hence, we assume the following protocol for $\alpha \gg (\Psi_1, ..., \Psi_n)$: at some time moment t and some $1 \leq i \leq n$, master decides to issue the command

$$DO(\Psi_i).$$

Then, at some later moment t', slave is expected to explicitly report an execution of this command:

$$DONE(\Psi_i).$$

The transfer from α to the effect of Φ_i is assumed to take place at some moment between t and t'.

For potential real-time applications, we may want to introduce a deadline parameter for $\gg$:

$$\alpha \gg^t (\Phi_1, ..., \Phi_n).$$

This means that at most time t should elapse between "DO" and "DONE". Another operator for which we might want to introduce a real-time parameter is $\rhd$: $\alpha \rhd^t \beta$ is a process which is true on an interval (i, j) iff there is e with $i < e \leq i + t < j$ such that α is true on (i, e) and β is true on (e, j). We leave exploring this possibility for the future, and the formal definitions of our language and semantics the reader will find in the later sections deal only with the non-real-time version.[2]

1.5 Our logic as a planning logic

Later sections contain examples showing how our logic can be used as a planning logic. A planning problem is represented as $\Delta \ddot{\rightarrow} \Gamma$, where Γ is a specification of the goal as a task, and Δ is the conjunction of the resources we possess. An action is understood as giving a command to one of these resources or, — at a higher level of abstraction, — assigning one resource to another, conditional, resource. Hence, actions change only those (sub)resources to which they are applied. The effect of an action for the rest of the resources is "no change", and it is this property that makes the logic frame-problem-free. Some examples also show how our logic can naturally handle certain planning problems which, under the traditional approach, would require special means for representing knowledge.

[2]The reader will also find that the language which we described here is a slightly simplified version of our real formalism.

2 Facts

The components of our language, shared by all three types of expressions (facts, processes and resources), are *variables* and *constants*. The set of variables is infinite. The set of constants may be finite or infinite. For now, we will make a simplifying assumption that the set of constants is $\{0, 1, 2, ...\}$. The set of *terms* is the union of the set of variables and the set of constants.

We also have a set of *fact letters* (called predicate letters in classical logic), with each of which is associated a natural number called *arity*.

Facts are the elements of the smallest set F of expressions, such that, saying "A is a fact" for "$A \in F$", we have:

- $\perp$ is a fact;
- if P is an n-ary fact letter and $t_1, ..., t_n$ are terms, then $P(t_1, ..., t_n)$ is a fact;
- if A and B are facts, then $(A)\rightarrow(B)$ is a fact;
- if A is a fact and x is a variable, then $\forall x(A)$ is a fact.

As we see, facts are nothing but formulas of classical first order logic. In the sequel, we will often omit some parentheses when we believe that this does not lead to ambiguity.

The other classical operators are introduced as standard abbreviations:

- $\neg A = A\rightarrow\perp$;
- $A\vee B = (\neg A)\rightarrow B$;
- $A\wedge B = \neg(\neg A\vee\neg B)$;
- $A\leftrightarrow B = (A\rightarrow B)\wedge(B\rightarrow A)$;
- $\top = \neg\perp$;
- $\exists xA = \neg\forall x\neg A$.

A *free variable* of a fact is a variable x which has an occurrence in the fact which is not in the scope of $\forall x$ (not *bound* by $\forall$). A fact is *closed*, if it has no free variables.

A *situation* is a set s of closed facts such that, using the expression $s \models A$ for $A \in s$, we have:

- $s \not\models \perp$;
- $s \models A\rightarrow B$ iff $s \not\models A$ or $s \models B$;
- $s \models \forall xA(x)$ iff $s \models A(a)$ for every constant a.

If $s \models A$, we say that A is *true*, or *holds* in situation s.

We fix an infinite set $\mathcal{T}$ of *time moments* together with a strict linear ordering relation $<$ on it. $i \leq j$, as one can expect, means $i < j$ *or* $i = j$. We assume that $0 \in \mathcal{T}$ and, for all $i \in \mathcal{T},\ \ 0 \leq i$.

$\mathcal{T}^+$ denotes the set $\mathcal{T} \cup \{\infty\}$. The ordering relation $<$ is extended to $\mathcal{T}^+$ by assuming that for all $t \in \mathcal{T},\ \ t < \infty$.

An *interval* is a pair (i, j), where $i \in \mathcal{T}$, $j \in \mathcal{T}^+$ and $i < j$.

A *world* is a function W which assigns to every time moment $i \in \mathcal{T}$ a situation $W(i)$.

3 Processes

This section contains formal definitions for the syntax and the semantics of the process level of our language. The reader can be advised to go back to page 4 to refresh their memory regarding the intuition and the motivation behind the key process operators introduced below. Some later sections — first of all, Section 7, — contain examples that could also help the reader to better understand our formalism in action.

The same applies to the next section, where we give formal definitions for the resource level, the motivation and intuition being explained in the introductory section.

Definition 3.1 The set of *finitary processes* is the smallest set FP of expressions such that, saying "α is a finitary process" for "$\alpha \in FP$", we have:

1. if A is a fact, then $\angle A$ is a finitary process;
2. if A is a fact, then $\updownarrow A$ is a finitary process;
3. if A is a fact, then $\upuparrows A$ is a finitary process;
4. if A is a fact, then $\Box A$ is a finitary process;
5. if α and β are finitary processes, then so is $\alpha \dot{\rightarrow} \beta$;
6. if α is a finitary process and x is a variable, then $\dot{\forall} x \alpha$ is a finitary process;
7. if α and β are finitary processes, then so is $\alpha \rhd \beta$;
8. if α is a finitary process, then so is $[\rhd]\alpha$;
9. if α is a finitary process, then so is $[\unrhd]\alpha$.

Some other process operators are introduced as abbreviations:

- $\dot{\bot} = \Box \bot$;
- $\dot{\top} = \angle \top$;
- $\dot{\neg}\alpha = \alpha \dot{\rightarrow} \dot{\bot}$;
- $\alpha \dot{\vee} \beta = (\dot{\neg}\alpha) \dot{\rightarrow} \beta$;
- $\alpha \dot{\wedge} \beta = \dot{\neg}(\dot{\neg}\alpha \dot{\vee} \dot{\neg}\beta)$;
- $\alpha \dot{\leftrightarrow} \beta = (\alpha \dot{\rightarrow} \beta) \dot{\wedge} (\beta \dot{\rightarrow} \alpha)$;
- $\dot{\exists} x \alpha = \dot{\neg} \dot{\forall} x \dot{\neg} \alpha$;
- $\Updownarrow \alpha = \angle \alpha \dot{\wedge} \upuparrows \alpha$;
- $\downdownarrows \alpha = \angle \alpha \dot{\wedge} \updownarrow \alpha$;
- $\alpha \unrhd \beta = \alpha \dot{\vee} (\alpha \rhd \beta)$.

A *closed process* is a process in which every variable is bound by $\forall$ or $\dot{\forall}$.

Definition 3.2 *Truth* of a closed finitary process γ on an interval (i, j) in a world W, symbolically $W \models_{i,j} \gamma$, is defined as follows:

- $W \models_{i,j} \angle A$ iff $W(i) \models A$;
- $W \models_{i,j} \updownarrow A$ iff for all $r \in \mathcal{T}$ with $i < r < j$, $W(r) \models A$;

- $W \models_{i,j} \Uparrow A$ iff for all $r \in \mathcal{T}$ with $i < r \leq j$, $W(r) \models A$;
- $W \models_{i,j} \Box A$ iff for all $r \in \mathcal{T}$, $W(r) \models A$;
- $W \models_{i,j} \alpha \dot{\to} \beta$ iff $W \not\models_{i,j} \alpha$ or $W \models_{i,j} \beta$;
- $W \models_{i,j} \dot{\forall} x \alpha(x)$ iff for every constant a, $W \models_{i,j} A(a)$;
- $W \models_{i,j} \alpha \rhd \beta$ iff there is e with $i < e < j$ such that $W \models_{i,e} \alpha$ and $W \models_{e,j} \beta$;
- $W \models_{i,j} [\rhd]\alpha$ iff there are $e_0, e_1, e_2, ... \in \mathcal{T}$ with $e_0 = i$, $e_0 < e_1 < e_2 < ... < j$ such that for every k, $W \models_{e_k, e_{k+1}} \alpha$;
- $W \models_{i,j} [\unrhd]\alpha$ iff $W \models_{i,j} [\rhd]\alpha$ or there are $e_0, ..., e_n \in \mathcal{T}^+$ such that $e_0 = i$, $e_n = j$ and, for every $k:\ 0 \leq k < n$, $W \models_{e_k, e_{k+1}} \alpha$.

When we later define a semantics for resources, we will also have to deal with infinite process expressions.

Definition 3.3 An *infinitary process* is defined by replacing in the definition of finitary process the word "finitary" by "infinitary" and adding the following 3 clauses:

- if $\alpha_0, \alpha_1, \alpha_2, ...$ are infinitary processes, then so is $\alpha_0 \dot{\wedge} \alpha_1 \dot{\wedge} \alpha_2 \dot{\wedge} ...$;
- if $\alpha_0, \alpha_1, \alpha_2, ...$ are infinitary processes and $k_1, k'_1, k_2, k'_2, ..., \in \mathcal{T}$, $k_1 < k'_1 < k_2 < k'_2 < ...$, then $\alpha_0 \leadsto_{k_1}^{k'_1} \alpha_1 \leadsto_{k_2}^{k'_2} \alpha_2 ...$ is an infinitary process;
- if $\alpha_0, ..., \alpha_n$ $(n \geq 1)$ are infinitary processes and $k_1, k'_1, ..., k_n, k'_n \in \mathcal{T}$, $k_1 < k'_1 < ... < k_n < k'_n$, then $\alpha_0 \leadsto_{k_1}^{k'_1} ... \leadsto_{k_n}^{k'_n} \alpha_n$ is an infinitary process.

Thus, every finitary process is, at the same time, an infinitary process, but not vice versa. We will use the common name "process" for both types of processes.

Definition 3.4 To get the definition of *truth* for closed infinitary processes, we replace the word "finitary" by "infinitary" in Definition 3.2 and add the following 3 clauses:

- $W \models_{i,j} \alpha_0 \dot{\wedge} \alpha_1 \dot{\wedge} \alpha_2 \dot{\wedge} ...$ iff for every k, $W \models_{i,j} \alpha_k$;
- $W \models_{i,j} \alpha_0 \leadsto_{k_1}^{k'_1} \alpha_1 \leadsto_{k_2}^{k'_2} \alpha_2 ...$ iff there are $e_0, e_1, e_2, ... \in \mathcal{T}$ such that $e_0 = i$ and, for each $m \geq 1$ we have $k_m < e_m < k'_m$ and $W \models_{e_{m-1}, e_m} \alpha_{m-1}$.
- $W \models_{i,j} \alpha_0 \leadsto_{k_1}^{k'_1} ... \leadsto_{k_n}^{k'_n} \alpha_n$ iff there are $e_0, , ...e_{n+1} \in \mathcal{T}$ such that $e_0 = i$, $e_{n+1} = j$, for each $m:\ 1 \leq m \leq n$ we have $k_m < e_m < k'_m$ and, for each $m:\ 1 \leq m \leq n+1$ we have $W \models_{e_{m-1}, e_m} \alpha_{m-1}$.

Definition 3.5 A closed process is said to be *valid* iff it is true on every interval of every world.

Here are some simple observations (all processes are assumed to be closed):

All processes of the form of a classical tautology, such as, say, $\alpha \dot{\vee} \dot{\neg} \alpha$ or $\alpha \dot{\to} \alpha \dot{\wedge} \alpha$, are valid.

$(\Uparrow A \dot{\wedge} \Uparrow B) \dot{\leftrightarrow} \Uparrow (A \wedge B)$ is valid.

$(\angle A \dot{\vee} \angle B) \dot{\leftrightarrow} \angle(A \vee B)$ is valid.

$(\Uparrow A \dot{\vee} \Uparrow B) \dot{\to} \Uparrow (A \vee B)$ is valid, but $\Uparrow (A \vee B) \dot{\to} (\Uparrow A \dot{\vee} \Uparrow B)$ is not.

$\rhd$ is associative: $\big(\alpha \rhd (\beta \rhd \gamma)\big) \dot{\leftrightarrow} \big((\alpha \rhd \beta) \rhd \gamma\big)$ is valid.

4 Resources

At this level of the language we have 3 sorts of expressions: resources, DO-resources and DONE-resources. We define them simultaneously:

Definition 4.1

1. A *DO-resource* is an expression $\gg x_1, ..., x_m(\Phi_1, ..., \Phi_n)$ $(m \geq 0,\ n \geq 0)$, where the x_i are variables and the Φ_i are DONE-resources. If $n = 1$, we can omit the parentheses and write just $\gg x_1, ..., x_m\Phi$; if $n = 0$, we write just $\gg$.

2. A *DONE-resource* is an expression $\ll x_1, ..., x_m(\Phi_1, ..., \Phi_n)$ $(m \geq 0,\ n \geq 1)$, where the x_i are variables and the Φ_i are resources. If $n = 1$, we can omit the parentheses and write just $\ll x_1, ..., x_m\Phi$;

3. A *resource* is one of the following:
   - $\alpha\Phi$, where α is a finitary process and Φ is a DO-resource;
   - $\Phi \ddot{\rightarrow} \Psi$, where Φ, Ψ are resources;
   - $\Phi \ddot{\wedge} \Psi$, where Φ, Ψ are resources;[3]
   - $\ddot{\forall} x\Phi$, where x is a variable and Φ is a resource.

We also introduce the following abbreviations:

- $\ddot{\bot} = \dot{\bot} \gg$;
- $\ddot{\top} = \dot{\top} \gg$;
- $\ddot{\neg}\Phi = \Phi \ddot{\rightarrow} \ddot{\bot}$;
- $\Phi \ddot{\vee} \Psi = (\ddot{\neg}\Phi) \ddot{\rightarrow} \Psi$;
- $\ddot{\exists} x\Phi = \ddot{\neg}\ddot{\forall} x \ddot{\neg}\Phi$;
- $\Phi \overline{\wedge} \Psi = \dot{\top} \gg (\ll \Phi, \ll \Psi)$;
- $\overline{\forall} x\Phi = \dot{\top} \gg x \ll \Phi$;
- $\Phi \overline{\vee} \Psi = \ddot{\neg}(\ddot{\neg}\Phi \overline{\wedge} \ddot{\neg}\Psi)$;
- $\overline{\exists} x\Phi = \ddot{\neg}(\overline{\forall} x \ddot{\neg}\Phi)$.

Thus, every resource is a $(\ddot{\rightarrow}, \ddot{\wedge}, \ddot{\forall})$-combination of expressions of the type

$$\alpha \gg \vec{x}\Big(\ll \vec{y_1}\big(\Phi^1_1(\vec{x}, \vec{y_1}), \ldots, \Phi^1_{k_1}(\vec{x}, \vec{y_1})\big), \ldots, \ll \vec{y_n}\big(\Phi^n_1(\vec{x}, \vec{y_n}), \ldots, \Phi^n_{k_n}(\vec{x}, \vec{y_n})\big)\Big),$$

where α is a finitary process and the Φ^j_i are resources. This expression represents an agent which maintains the process α as its effect; a command to it should be given by specifying $\vec{x}$ as a sequence $\vec{a}$ of constants, and specifying one of the i, $1 \leq i \leq n$. The expression to the right of $\gg$ represents the potential of this resource. We see that this potential is more complex than the type of potentials discussed in Section 1. The intuitive meaning of $\ll \vec{y_i}\big(\Phi^i_1(\vec{a}, \vec{y_i}), \ldots, \Phi^i_{k_i}(\vec{a}, \vec{y_i})\big)$ as a command is that slave has to produce the resource $\Phi^i_j(\vec{a}, \vec{b})$ for a j $(1 \leq j \leq k_i)$ and $\vec{b}$ of his own choice, and report about this choice (along with the fact of executing the command) to master.

[3] In fact, $\Phi \ddot{\wedge} \Psi$ can be defined as $\ddot{\neg}(\ddot{\neg}\Phi \ddot{\vee} \ddot{\neg}\Psi)$ (see the $\ddot{\wedge}$-independent definition of $\ddot{\neg}$ and $\ddot{\vee}$ below), but we still prefer to treat $\ddot{\wedge}$ as a basic symbol.

The operators $\gg$ and $\ll$, when followed by a nonempty list of variables, act as quantifiers: they bind the occurrences of these variables within their scope. An occurrence of a variable in a resource is said to be *free*, if it is not in the scope of $\gg$, $\ll$, $\forall$, $\dot{\forall}$ or $\ddot{\forall}$. If a resource does not contain free occurrences of variables, then it is said to be *closed*.

Note that our definition of resource allows infinite expressions: there is no requirement "... is the smallest set of expressions such that...". Naturally, we only want to deal with resources which can be expressed finitarily. One way to express a certain class of infinite resources finitarily is to allow *recursive definitions* for their subexpressions. For safety, we will only allow definitions that have the following form:

$$\Phi := \gg \vec{x}\Big(\ll \vec{y_1}(\alpha_1^1\Phi_1^1,\ldots,\alpha_{k_1}^1\Phi_{k_1}^1),\ldots,\ll \vec{y_n}(\alpha_1^n\Phi_1^n,\ldots,\alpha_{k_n}^n\Phi_{k_n}^n)\Big), \tag{1}$$

where the α_i^j are finitary processes and the Φ_i^j are DO-resources introduced by the same type (1) of recursive definitions, so that, Φ, itself, can be among the Φ_i^j.

A recursive definition is not a part of a formula but should be given separately, and if a resource contains a recursively defined subexpression Φ, we assume that Φ just abbreviates the right-hand side of its definition.

Another type of finitarily represented infinite expressions we will allow in resources is

$$!\Phi,$$

which is understood as an abbreviation for the infinite conjunction

$$\Phi\ddot{\wedge}\Phi\ddot{\wedge}\Phi\ddot{\wedge}...$$

We will call the resources that are finite expressions, possibly containing !-expressions and recursively defined subexpressions of the form (1), *finitary resources*. Since we are going to deal only with this type of resources, from now on, the word "resource" will always refer to finitary resources.

We are now going to give a formal definition of the semantics for resources. This definition is in a game-semantical style as we understand a resource as a potential game between master and slave, where moves consist in giving commands and/or reporting execution of commands.

A *position* is one of the following:

- a resource;
- a DONE-resource;
- $\Phi\ddot{\rightarrow}\Psi$, where Φ and Ψ are positions;
- $\Phi\ddot{\wedge}\Psi$, where Φ and Ψ are positions;
- $\ddot{\forall}x\Phi$, where x is a variable and Φ is a position.

When speaking about a subexpression of an expression, we are often interested in a concrete occurrence of this subexpression rather than the subexpression as an expression (which may have several occurrences). In order to stress that we mean a concrete occurrence, we shall use the words "osubexpression", "osubposition", etc. ("o" for "occurrence").

A *surface osubexpression* of a resource or a position is an osubexpression which is not in the scope of $\gg$ or $\ll$.

Such an osubexpression is *positive*, or has a *positive occurrence*, if it is in the scope of an even number of $\ddot{\rightarrow}$; otherwise it is *negative*.

Definition 4.2 A *master's move* for a position Φ is a position which results from Φ by

- replacing some finite (possibly zero) number of positive surface osubpositions of the form $\alpha \gg \vec{x}\big(\Psi_1(\vec{x}),...,\Psi_n(\vec{x})\big)$ by $\Psi_i(\vec{a})$ for some sequence $\vec{a}$ (of the same length as $\vec{x}$) of constants and some i $(1 \leq i \leq n)$, and/or
- replacing some finite (possibly zero) number of negative surface osubpositions of the form $\ll \vec{x}\big(\Psi_1(\vec{x}),...,\Psi_n(\vec{x})\big)$ by $\Psi_i(\vec{a})$ for some $\vec{a}$ and i $(1 \leq i \leq n)$.

Slave's move is defined in the same way, only with the words "positive" and "negative" interchanged.

Thus, master's moves consist in giving commands in positive osubresources and reporting execution of commands in negative osubresources, while slave's moves consist in giving commands in negative osubresources and reporting execution of commands in positive osubresources.

Suppose Ψ' and Ψ'' are master's and slave's moves for a position Φ. Then the *composition* of these two moves with respect to Φ is the position Ψ which results from Φ by combining all the changes made by master and slave in Φ in their Φ-to-Ψ' and Φ-to-Ψ'' moves. Ψ is said to be a *move* for Φ. Note that every position is a move for itself.

For a position Φ, a Φ-*play*, or a *play over* Φ, is a finite or infinite sequence of the type

$$\langle \Phi_0, t_1, \Phi_1, t_2, \Phi_2, ...\rangle,$$

where $\Phi_0 = \Phi$, the t_i are increasing time moments $(t_1 < t_2 < ...)$ and, for each i, Φ_{i+1} is a move for Φ_i.

A play is said to be *compact*, if no two neighboring positions (Φ_i and Φ_{i+1}) in it are identical. If a play P is not compact, its *compactization*, denoted by P^+, is the play which results from P by deleting every position which is identical to the previous position, together with the time moment separating these two positions.

Intuitively, the Φ_i are the consecutive positions of the play, and t_i is the time moment at which the position Φ_{i-1} is replaced by Φ_i.

Note that the $(\ddot{\to}, \ddot{\wedge}, \ddot{\forall})$-structure of a position in a play is inherited by the subsequent positions.

Every (compact) play P produces a unique process P^* defined below. In this definition, "..." does not necessarily mean an infinite continuation of the list (play): such a list can be 1-element, n-element or infinite; in clause 6, $\vec{Q}$ stands for an arbitrary (possibly empty) sequence $t_1, \Gamma_1, t_2, \Gamma_2, ...$

Definition 4.3 (of the operation *)

1. $\langle \alpha \gg \Phi\rangle^* \ = \ \alpha.$
2. $\langle \Phi_0 \ddot{\to} \Psi_0,\ t_1,\ \Phi_1 \ddot{\to} \Psi_1,\ t_2,\ ...\rangle^* \ =$
   $\langle \Phi_0,\ t_1,\ \Phi_1,\ t_2,\ ...\rangle^{+*} \dot{\to} \langle \Psi_0,\ t_1,\ \Psi_1,\ t_2,\ ...\rangle^{+*}.$
3. $\langle \Phi_0 \ddot{\wedge} \Psi_0,\ t_1,\ \Phi_1 \ddot{\wedge} \Psi_1,\ t_2,\ ...\rangle^* \ =$
   $\langle \Phi_0,\ t_1,\ \Phi_1,\ t_2,\ ...\rangle^{+*} \dot{\wedge} \langle \Psi_0,\ t_1,\ \Psi_1,\ t_2,\ ...\rangle^{+*}.$
4. $\langle \ddot{\forall} x \Phi_0,\ t_1,\ \ddot{\forall} x \Phi_1,\ t_2,\ ...\rangle^* \ = \dot{\forall} x(\langle \Phi_0,\ t_1,\ \Phi_1,\ t_2,\ ..\rangle^*).$
5. $\langle \alpha \gg \vec{x}(\Psi_1(\vec{x}),...,\Psi_n(\vec{x})), t, \Psi_i(\vec{a})\rangle^* = \dot{\bot}.$

6. If $P = \langle$

$$\alpha \gg \vec{x}\Big(\ll \vec{y_1}\big(\Psi^1_1(\vec{x},\vec{y_1}),...,\Psi^1_{k_1}(\vec{x},\vec{y_1})\big),...,\ll \vec{y_n}\big(\Psi^n_1(\vec{x},\vec{y_n}),...,\Psi^n_{k_n}(\vec{x},\vec{y_n})\big)\Big),$$
$$k,$$
$$\ll \vec{y_i}\big(\Psi^i_1(\vec{a},\vec{y_i}),...,\Psi^i_{k_i}(\vec{a},\vec{y_i})\big),$$
$$m,$$
$$\Psi^i_j(\vec{a},\vec{b}),$$
$$\vec{Q}\rangle,$$

then

$$P^* \;=\; \alpha \rightsquigarrow^m_k \langle \Psi^i_j(\vec{a},\vec{b}),\vec{Q}\rangle^*.$$

Explanation: According to clause 3, a play over a $\ddot{\wedge}$-conjunction of resources produces the $\dot{\wedge}$-conjunction of the processes produced by the (sub)plays over the conjuncts of the resource. Similarly for the other double-dotted connectives $\ddot{\rightarrow}$ (clause 2) and $\ddot{\forall}$ (clause 4).

A play over

$$\alpha \gg \vec{x}\Big(\ll \vec{y_1}\big(\Psi^1_1(\vec{x},\vec{y_1}),...,\Psi^1_{k_1}(\vec{x},\vec{y_1})\big),...,\ll \vec{y_n}\big(\Psi^n_1(\vec{x},\vec{y_n}),...,\Psi^n_{k_n}(\vec{x},\vec{y_n})\big)\Big)$$

produces α, if no moves have been made (clause 1). If a command

$$\ll \vec{y_i}\big(\Psi^i_1(\vec{a},\vec{y_i}),...,\Psi^i_{k_i}(\vec{a},\vec{y_i})\big)$$

was given but a report never followed (clause 5), we consider this a failure of the non-reporting resource to carry out its task, and associate the always-false process $\dot{\perp}$ with this play so that it is never successful. Finally, if a report $\Psi^i_j(\vec{a},\vec{b})$ followed the command, the play produces $\alpha \rightsquigarrow^m_k \beta$, where k is the moment of giving the command, m is the moment of reporting its execution, and β is the process produced by the subplay over $\Psi^i_j(\vec{a},\vec{b})$; truth of this $\rightsquigarrow$-process means that the process α switches to the process β at some time after the command and before the report.

One can show that as long as Φ is a closed finitary process, the process P^* produced by a Φ-play P is always a closed infinitary process in the sense of Definition 3.3.

A *slave's strategy* is a function f which assigns to every position Φ a slave's move for Φ. We assume that this function is implemented as a program on a machine, and we denote by $f'(\Phi)$ the time this program takes to give an output for input Φ; if the program doesn't give any output, or gives an output which is not a slave's move for Φ, then we assume that $f'(\Phi) = \infty$.

Let Φ_0 be a resource and f be a slave's strategy. Here is an informal definition of a Φ_0-*play with slave's strategy* f. The play starts at moment 0, and at this stage it is the one-position (sub)play $\langle\Phi_0\rangle$. Slave, i.e. the function f, takes Φ_0 as an input, and starts computing an output for it, — thinking what move to make for Φ_0. While slave is thinking, master can make some moves $\Phi_1,...,\Phi_n$ at time moments $t_1,...,t_n$, where $n \geq 0$, $t_1 < ... < t_n$ and each Φ_i $(1 \leq i \leq n)$ is a master's move for Φ_{i-1}. Note that Φ_n is a master's move for Φ_0 by the transitivity of this relation. The play has thus evolved to

$$\langle\Phi_0, t_1, \Phi_1, ..., t_n, \Phi_n\rangle.$$

Finally, at moment $t_{n+1} = f'(\Phi_0)$, f computes a slave's move Ψ for Φ_0, and the next two items of the play become t_{n+1} and Φ_{n+1}, where Φ_{n+1} is the composition of Ψ and Φ_n with respect to Φ_0. Note that Φ_{n+1} is, at the same time, a slave's move for Φ_n.

So far slave has been busy processing the input Φ_0 and did not see master's moves. Only now he looks at the current (last) position and sees that it is Φ_{n+1}. So, he takes this position as a new input, and starts computing a move for it. While slave is thinking on his second move, master can continue making moves and the play can evolve to

$$\langle \Phi_0, t_1, \Phi_1, ..., t_n, \Phi_n, t_{n+1}, \Phi_{n+1}, ..., t_m, \Phi_m \rangle$$

until, at some moment t_{m+1}, slave comes up with a move Γ for Φ_{n+1}. The next two items of the play become t_{m+1} and Φ_{m+1}, where Φ_{m+1} is the composition of Γ and Φ_m with respect to Φ_{n+1}. And so on...

If, at some stage, f fails to compute a move, that is, thinks for an infinitely long time, then all the further moves will be made only by master. In this case, master may make not only a finite, but also an infinite number of consecutive moves.

We say that a play P is *successful* with respect to a world W, iff $W \models_{0,\infty} P^*$.

A slave's strategy is said to be *universally successful* for a closed resource Φ, iff every Φ-play with this strategy is successful with respect to every world.

Definition 4.4 We say that a resource is *universally valid* iff there is a universally successful slave's strategy for it.

5 Resource schemata

In this section we extend our language by adding to it *resource letters.* Formulas of this extended language can be viewed as schemata for resources, where resource letters stand for resources.

Every resource letter has a fixed arity. The definition of *resource scheme* is the same as the definition of resource (where the word "resource" is replaced by "resource scheme"), with the following additional clause:

- if Φ is an n-ary resource letter and $t_1, ..., t_n$ are terms, then $\Phi(t_1, ..., t_n)$ is a resource scheme.

For safety, we assume that the set of variables occurring in resource schemata is a proper subset of the set of variables of the language introduced in the previous sections, and that there are infinitely many variables in the latter that don't occur in resource schemata.

A resource is said to be *safe* if it is $\dot{\top}\Phi$ for some DO-resource Φ, or a $(\ddot{\to}, \ddot{\wedge}, \ddot{\forall})$-combination of resources of this type.

Safe resources are what we could call "effectless" resources: they are not responsible for maintaining any nontrivial process and their value is associated only with their convertibility into other resources.

A *substitution* (resp. *safe substitution*) is a function τ which assigns to every n-ary resource letter Φ a resource (resp. safe resource) $\tau\Phi = \Psi(x_1, ..., x_n)$ with exactly n free variables which does not contain any variables that might occur in resource schemata.

Given a resource scheme Φ and a substitution τ, Φ^τ is defined as a result of substituting in Φ every resource letter P by τP. More precisely,

- for an atomic resource scheme Φ of the form $P(t_1, ..., t_n)$, where the t_i are terms and $\tau P = \Psi(x_1, ..., x_n)$, we have $\Phi^\tau = \Psi(t_1, ..., t_n)$;

- $\Big(\alpha \gg \vec{x}\Big(\ll \vec{y_1}(\Phi^1_1, \ldots, \Phi^1_{k_1}), \ldots, \ll \vec{y_n}(\Phi^n_1, \ldots, \Phi^n_{k_n})\Big)\Big)^\tau =$
  $\alpha \gg \vec{x}\Big(\ll \vec{y_1}((\Phi^1_1)^\tau, \ldots, (\Phi^1_{k_1})^\tau), \ldots, \ll \vec{y_n}((\Phi^n_1)^\tau, \ldots, (\Phi^n_{k_n})^\tau)\Big)$;

- $(\Phi \ddot{\to} \Psi)^\tau = \Phi^\tau \ddot{\to} \Psi^\tau$;

- $(\Phi \ddot{\wedge} \Psi)^\tau = \Phi^\tau \ddot{\wedge} \Psi^\tau$;
- $(\ddot{\forall} x \Phi)^\tau = \ddot{\forall} x(\Phi^\tau)$.

We say that a resource Φ is an *instance* of a resource scheme Ψ, iff $\Phi = \Psi^\tau$ for some substitution τ. If τ is a safe substitution, then Φ is said to be a *safe instance* of Ψ.

Definition 5.1 We say that a resource scheme Ψ is *universally valid* (resp. *universally s-valid*) iff there is slave's strategy such that for every instance (resp. safe instance) Φ of Ψ, the Φ-play with this strategy is successful with respect to every world.

6 The MALL and MLL fragments

Our logic, — the set of universally valid resources or resource schemata, — is certainly undecidable in the full language as it contains first order classical logic. However, some reasonably efficient heuristic algorithms can apparently be found for it. Also, some natural fragments of the logic are decidable. This paper doesn't address these issues in detail as its main goal is to introduce the language and the semantics and show possible applications in case efficient algorithms are elaborated. This is a beginning of the work rather than a completed work.

Here we only state the decidability of two fragments of the logic. The first one we call the MALL fragment. Its language is the same as that of Multiplicative-Additive Linear Logic, where $\ddot{\rightarrow}$, $\ddot{\wedge}$, $\overline{\wedge}$ and $\overline{\forall}$ correspond to the multiplicative implication, multiplicative conjunction, additive conjunction and (additive) universal quantifier of linear logic, respectively. Here is the definition:

MALL-formulas are the elements of the smallest class M of expressions such that, saying "Φ is a MALL-formula" for $\Psi \in M$, we have:

- $\ddot{\bot}$ is a MALL-formula;
- if Ψ is an n-ary resource letter and $t_1, ..., t_n$ are terms, then $\Psi(t_1, ..., t_n)$ is a MALL-formula;
- if Ψ and Φ are MALL-formulas, then so is $\Phi \ddot{\rightarrow} \Psi$;
- if Ψ and Φ are MALL-formulas, then so is $\Phi \ddot{\wedge} \Psi$;
- if Ψ and Φ are MALL-formulas, then so is $\Phi \overline{\wedge} \Psi$;
- if Ψ is a MALL-formula and x is a variable, then $\overline{\forall} x \Psi$ is a MALL-formula.

Here is our main technical claim:

Claim 6.1 *The set of universally s-valid closed MALL formulas is decidable. In particular, it is the logic ET introduced in [8]. The decision algorithm is constructive: it not only states the existence of a successful strategy (when it exists), but actually finds such a strategy.*

We let this claim go without a proof. An interested reader who carefully studies the relevant parts of [8] should be able to re-write the soundness and completeness proof for *ET* given there as a proof of the above claim. In fact, the proof given there establishes the completeness of *ET* in a much stronger sense than claimed above.

A *MLL-formula* ("Multiplicative Linear Logic") is a MALL-formula which does not contain $\overline{\wedge}$ or $\overline{\forall}$. Since we have no quantifiers, we assume that all resource letters in MLL-formulas are 0-ary. We have a stronger soundness/decidability result for the MLL-fragment of our logic. Stronger in the sense that it is about validity rather than s-validity.

A MLL-formula is said to be a *binary tautology*, if it is an instance of a classical tautology (with the double-dots placed over $\perp$, $\wedge$ and $\rightarrow$) in which every predicate letter (non-$\perp$ propositional atom) occurs at most twice. For example, $\Phi\ddot{\wedge}\Psi\ddot{\rightarrow}\Phi$ is a binary tautology, and so is $\Phi\ddot{\wedge}\Phi\ddot{\rightarrow}\Phi$ as the latter is an instance of the former; however, $\Phi\ddot{\rightarrow}\Phi\ddot{\wedge}\Phi$ is not a binary tautology. Note that in fact a binary tautology is always an instance of a classical tautology where every predicate letter has either one occurrence, or two occurrences, one of which is positive and the other — negative. Blass [2] was the first to study binary tautologies and find a natural semantics for them.

Claim 6.2 *A MLL-formula is universally valid iff it is a binary tautology. Hence, validity for MLL-formulas is decidable; again, the decision algorithm is constructive.*

The "only if" part of this claim follows from Claim 6.1 together with an observation that a MLL-formula is a binary tautology iff it is in ET. The "if" part, as always, is easier to verify, and instead of giving an actual proof, we will just explain the idea behind it on particular examples.

The simplest binary tautology is $\Phi\ddot{\rightarrow}\Phi$. Why is it universally valid? Observe that since one of the two occurrences of Φ is negative and the other occurrence is positive, what is a master's move in one occurrence of Φ, is a slave's move in the other occurrence of Φ, and vice versa. The slave's strategy which ensures that every play is successful, consists in *pairing* these two occurrences: copying master's moves, made in either occurrence, into the other occurrence. For example, let Φ be

$$\alpha \gg \Big(\ll (\beta \gg,\ \gamma \gg),\ \ \ll (\delta \gg)\Big).$$

Then, the initial position is

$$\alpha \gg \Big(\ll (\beta \gg,\ \gamma \gg),\ \ \ll (\delta \gg)\Big)\ \ddot{\rightarrow}\ \alpha \gg \Big(\ll (\beta \gg,\ \gamma \gg),\ \ \ll (\delta \gg)\Big).$$

Slave waits (keeps returning the above position without changes) until master makes a move. If master never makes a move, then (after compactization) we deal with a one-position (0-move) play and, according to the clauses 2 and 1 of Definition 4.3, the process produced by this play is $\alpha\dot{\rightarrow}\alpha$. Clearly, this process is valid and hence the play is successful. Otherwise, if master makes a move at some moment t_1, this should be replacing the positive occurrence of $\alpha \gg \Big(\ll (\beta \gg, \gamma \gg),\ \ll (\delta \gg)\Big)$ by either $\ll (\beta \gg, \gamma \gg)$ or $\ll (\delta \gg)$. Suppose the former. Thus, the next position of the play is

$$\alpha \gg (\ll (\beta \gg,\ \gamma \gg),\ \ \ll (\delta \gg))\ \ \ddot{\rightarrow}\ \ \ll (\beta \gg,\ \gamma \gg).$$

Then slave does the same in the antecedent of this position, thus making

$$\ll (\beta \gg,\ \gamma \gg)\ \ \ddot{\rightarrow}\ \ \ll (\beta \gg,\ \gamma \gg)$$

the next position of the play. This move will happen at time moment t_2 which is greater than t_1 by the time slave needs to copy the master's move.

After this, slave waits again until master reports an execution of this command. If this never happens, then the play is successful because, by the clauses 2 and 5 of Definition 4.3, the process it produces is $\dot{\perp}\dot{\rightarrow}\dot{\perp}$, which is always true. Otherwise, at some moment $t_3 > t_2$, master reports an execution by replacing $\ll (\beta \gg,\ \gamma \gg)$ by, say, $\gamma \gg$. So that the next position now becomes

$$\gamma \gg\ \ddot{\rightarrow}\ \ll (\beta \gg, \gamma \gg).$$

Then, as soon as he can, — at some moment t_4, — slave reports the same in the consequent of this position, and we get

$$\gamma \gg\ \ddot{\rightarrow} \gamma \gg .$$

Since there are no moves for this position, the play ends here. An analysis of the clause 6 of Definition 4.3 convinces us that the process produced by this play is

$$\left(\alpha \leadsto_{t_2}^{t_3} \gamma\right) \dot{\rightarrow} \left(\alpha \leadsto_{t_1}^{t_4} \gamma\right).$$

Since $t_1 < t_2 < t_3 < t_4$, it is easily seen that this process is valid, and thus the play is successful.

A similar strategy can be used for other binary tautologies. E.g., the strategy for $\Phi\ddot{\wedge}(\Phi\ddot{\rightarrow}\Psi) \ddot{\rightarrow} \Psi$ consists in pairing the two occurrences of Φ and pairing the two occurrences of Ψ. This trick, however, fails for resources that are not binary tautologies. For example, in $\Phi\ddot{\rightarrow}\Phi\ddot{\wedge}\Phi$, slave can pair the negative occurrence of Φ only with one of the two positive occurrences of Φ. The (sub)play over the unpaired occurrence then may produce a false process while the (sub)play over the negative occurrence — a true process. In that case the process produced by the whole play over $\Phi\ddot{\rightarrow}\Phi\ddot{\wedge}\Phi$ will be false.

7 The assembly world in terms processes

What is going on in a computer at the assembly language (as well as higher) level can be formalized in our language as a process. Here we consider an example of a simplified assembly world.

Our "computer" has only 3 memory locations or registers: L_1, initially containing 2, L_2, initially containing 0, and L_3, initially containing 0. The assembly language for this "architecture" has only 3 commands: command #1, command #2, command #3. Command #i results in adding the contents of the other two locations and writing it in L_i.

There are several ways to formalize this situation in our language, and here is one of them.

Since giving a command is an instantaneous event, we assume that command #i creates a "puls" which makes the contents of L_i change. Creating a puls means making a certain fact — denote it by P_i for command #i — become true for one moment. Between commands another fact, Np ("no puls"), holds. It can be defined by

$$Np =_{df} \neg\left(P_1 \vee P_2 \vee P_3\right). \tag{2}$$

The further abbreviations we will use for convenience are:

$$\begin{aligned}
\alpha_1 &=_{df} \ \angle P_1 \dot{\wedge} \updownarrow Np \\
\alpha_2 &=_{df} \ \angle P_2 \dot{\wedge} \updownarrow Np \\
\alpha_3 &=_{df} \ \angle P_3 \dot{\wedge} \updownarrow Np
\end{aligned}$$

Thus, issuing command #i can be seen as starting (switching to) process α_i.

The formulas λ_1, λ_2 and λ_3, defined below, are meant to describe the behavior of the processes going on in the 3 locations:[4]

$$\begin{aligned}
\lambda_1 &=_{df} \ [\trianglerighteq]\Big(\angle P_1 \ \dot{\wedge} \ \updownarrow \neg P_1 \ \dot{\wedge} \ \dot{\exists} x, y\big(\angle L2(x) \dot{\wedge} \angle L3(y) \dot{\wedge} \Uparrow L1(x+y)\big)\Big) \\
\lambda_2 &=_{df} \ [\trianglerighteq]\Big(\angle P_2 \ \dot{\wedge} \ \updownarrow \neg P_2 \ \dot{\wedge} \ \dot{\exists} x, y\big(\angle L1(x) \dot{\wedge} \angle L3(y) \dot{\wedge} \Uparrow L2(x+y)\big)\Big) \\
\lambda_3 &=_{df} \ [\trianglerighteq]\Big(\angle P_3 \ \dot{\wedge} \ \updownarrow \neg P_3 \ \dot{\wedge} \ \dot{\exists} x, y\big(\angle L1(x) \dot{\wedge} \angle L2(y) \dot{\wedge} \Uparrow L3(x+y)\big)\Big)
\end{aligned}$$

[4]Although our language does not allow terms such as $x + y$, we can pretend that it does, because every expression containing this kind of terms can be rewritten as an equivalent legal expression the language defined in the previous sections. So that, for convenience, here and later we assume that our language is based on predicate logic with function symbols and identity rather than pure predicate logic.

Before we analyze the λ_i, let us agree on some jargon. Every (true) process $[\rhd]\gamma$ or $\gamma_1 \rhd ... \rhd \gamma_n$ can be divided into $\rhd$-*stages*, which are the consecutive time intervals on which γ is true. A transition from one stage to another will be referred to as a $\rhd$-*transition*. Similarly, we will use the terms "$\unrhd$-stage" and "$\unrhd$-transition" for processes of the type $[\unrhd]\gamma$ or $\gamma_1 \unrhd ... \unrhd \gamma_n$.

In these terms, λ_i starts its $\unrhd$-stage when puls P_i is given (the conjunct $\angle P_i$), and will stay in this stage exactly until the same puls is given again. A $\unrhd$-transition to the new stage before this is impossible because, due to the conjunct $\angle P_i$, that stage requires that P_i be true at the moment of the transition. And a late transition to a new stage is also impossible because, as soon as P_i becomes true, the conjunct $\Uparrow \neg P_i$ is violated. Throughout each $\unrhd$-stage, except its first moment, the location L_i then stores the sum of the values that the other two locations had at the initial moment of the stage.

Now we need to axiomatize the situation where the initial value of $L1$ is 2, the initial values of the other two locations are 0, and these values will be maintained until the corresponding command (puls) is given. This will be captured by the following 3 axioms:

$$\Big((\Updownarrow L_1(2))\dot{\wedge}(\Downarrow \neg P_1)\Big) \unrhd \lambda_1 \tag{3}$$

$$\Big((\Updownarrow L_2(0))\dot{\wedge}(\Downarrow \neg P_2)\Big) \unrhd \lambda_2 \tag{4}$$

$$\Big((\Updownarrow L_3(0))\dot{\wedge}(\Downarrow \neg P_3)\Big) \unrhd \lambda_3 \tag{5}$$

Next, for safety, we need to state that two different pulses cannot happen simultaneously:

$$\Box\Big(\neg(P_1 \wedge P_2)\wedge\neg(P_1 \wedge P_3)\wedge\neg(P_2 \wedge P_3)\Big) \tag{6}$$

We also need to axiomatize some sufficient amount of the arithmetic needed. We may assume that *Arithm* is the conjunction of the axioms of Robinson's arithmetic (see [11]), although, for our purposes, just

$$2 + 0 = 2 \wedge 2 + 2 = 4 \wedge 2 + 4 = 6 \wedge 6 + 4 = 10$$

would do as *Arithm*. In any case,

$$\Box Arithm \tag{7}$$

should be one of our axioms.

The final axiom below represents a program which, after being idle ($\Downarrow Np$), issues command #2, then command #3, then command #1 and then, again, command #2.

$$(\Downarrow Np) \rhd \alpha_2 \rhd \alpha_3 \rhd \alpha_1 \rhd \alpha_2. \tag{8}$$

Our claim is that given the truth of these axioms, we can conclude that the process $\Updownarrow L_2(10)$ will be reached at some point. In other words, the process

$$\Big((3)\dot{\wedge}(4)\dot{\wedge}(5)\dot{\wedge}(6)\dot{\wedge}(7)\dot{\wedge}(8)\Big)\dot{\rightarrow}\Big(\dot{\top}\rhd \Updownarrow L_2(10)\Big)$$

is valid. Indeed, in the initial situation,[5] we have

$$L_1(2),\ L_2(0),\ L_3(0),\ Np.$$

[5]We use the word "situation" with a relaxed meaning: here it denotes some "core" subset of facts rather than a complete set of facts.

While we have Np, the values of L_1, L_2, L_3 cannot change, because a $\trianglerighteq$-transition to λ_i would require the truth (at the moment of transition) of P_i, which is ruled out by (2). So, the situation will change only if a puls P_i occurs.

The first stage $\Downarrow Np$ of axiom 8 prevents occurring such pulses. So, the situation will change exactly when the process (8) makes a $\triangleright$-transition from $\Downarrow Np$ to α_2, i.e. to $\angle P_2 \dot\wedge \updownarrow Np$. This transition forces (4) to switch to λ_2, which results in starting the process $\Uparrow L_2(2)$: L_2 will have its old value 0 at the first moment of the stage, and the value 2 after that. On the other hand, in view of (6), no $\trianglerighteq$-transition can happen in (3) or (5). Thus, the situation at the moment of transition becomes

$$L_1(2),\ L_2(0),\ L3(0),\ P_2,$$

which will become

$$L_1(2),\ L_2(2),\ L3(0),\ Np$$

right after the moment of transition because of $\Uparrow L_2(2)$ and $\updownarrow Np$.

Continuing arguing in this manner, we get that the further development of situations is:

$$L_1(2),\ L_2(2),\ L_3(0),\ P_3,$$

$$L_1(2),\ L_2(2),\ L_3(4),\ Np,$$

$$L_1(2),\ L_2(2),\ L_3(4),\ P_1,$$

$$L_1(6),\ L_2(2),\ L_3(4),\ Np,$$

$$L_1(6),\ L_2(2),\ L_3(4),\ P_2,$$

$$L_1(6),\ L_2(10),\ L_3(4),\ Np.$$

Since the last stage of the program (8) contains the conjunct $\updownarrow Np$, no further changes will occur, and the value of L_2 will remain 10.

8 The assembly world in terms of resources

In the previous example we dealt with processes that ran "by themselves". We could not interfere and manage them, so that there was no space for planning.

Presenting the world as a set of resources rather than processes allows us to capture our ability to influence the course of events in the world. Our way to interact with the world is giving and receiving commands.

Here is an attempt to present the assembly world as a resource. We assume that we have, as an empty-potential resource, the $\dot\wedge$-conjunction Γ of the axioms (3)-(7), suffixed by $\gg$:

$$\Gamma =_{df} \Big((3)\dot\wedge(4)\dot\wedge(5)\dot\wedge(6)\dot\wedge(7)\Big) \gg .$$

As for axiom (8), which is a "ready program", instead of it we have a resource which accepts from us any of those 3 commands, as many times as we like.

This resource will be expressed by

$$\big(\Downarrow Np\big)\Theta, \tag{9}$$

where Θ is introduced by the recursive definition

$$\Theta := \gg \Big(\ll (\alpha_1\Theta), \ll (\alpha_2\Theta), \ll (\alpha_3\Theta)\Big).$$

Now we can ask the question if we can accomplish the task $\Big(\dot\top \triangleright (\Updownarrow L2(10))\Big) \gg$. In other words, whether

$$\Gamma\ddot\wedge(9) \dot\to \Big(\Big(\dot\top \triangleright (\Updownarrow L2(10))\Big) \gg \Big)$$

is universally valid. Yes, it is. The strategy is:

Convert $(\Downarrow Np)\Theta$ into $\ll(\alpha_2\Theta)$; after getting the report $\alpha_2\Theta$, convert it into $\ll(\alpha_3\Theta)$; after getting the report $\alpha_3\Theta$, convert it into $\ll(\alpha_1\Theta)$; after getting the report $\alpha_1\Theta$, convert it into $\ll(\alpha_2\Theta)$, and stop.

Thus, unless (9) fails to carry out its task, the only play corresponding to this strategy is the following sequence of positions:

0. $\Gamma\ddot{\wedge}\big((\Downarrow Np)\Theta\big) \ddot{\to} \Big(\Big(\dot{\top} \rhd \big(\Updownarrow L2(10)\big)\Big) \gg\Big)$
1. $\Gamma\ddot{\wedge}\big(\ll\alpha_2\Theta\big) \ddot{\to} \Big(\Big(\dot{\top} \rhd \big(\Updownarrow L2(10)\big)\Big) \gg\Big)$

1′. $\Gamma\ddot{\wedge}\big(\alpha_2\Theta\big) \ddot{\to} \Big(\Big(\dot{\top} \rhd \big(\Updownarrow L2(10)\big)\Big) \gg\Big)$

2. $\Gamma\ddot{\wedge}\big(\ll\alpha_3\Theta\big) \ddot{\to} \Big(\Big(\dot{\top} \rhd \big(\Updownarrow L2(10)\big)\Big) \gg\Big)$

2′. $\Gamma\ddot{\wedge}\big(\alpha_3\Theta\big) \ddot{\to} \Big(\Big(\dot{\top} \rhd \big(\Updownarrow L2(10)\big)\Big) \gg\Big)$

3. $\Gamma\ddot{\wedge}\big(\ll\alpha_1\Theta\big) \ddot{\to} \Big(\Big(\dot{\top} \rhd \big(\Updownarrow L2(10)\big)\Big) \gg\Big)$

3′. $\Gamma\ddot{\wedge}\big(\alpha_1\Theta\big) \ddot{\to} \Big(\Big(\dot{\top} \rhd \big(\Updownarrow L2(10)\big)\Big) \gg\Big)$

4. $\Gamma\ddot{\wedge}\big(\ll\alpha_2\Theta\big) \ddot{\to} \Big(\Big(\dot{\top} \rhd \big(\Updownarrow L2(10)\big)\Big) \gg\Big)$

4′. $\Gamma\ddot{\wedge}\big(\alpha_2\Theta\big) \ddot{\to} \Big(\Big(\dot{\top} \rhd \big(\Updownarrow L2(10)\big)\Big) \gg\Big)$.

One can easily see that this play produces the same process as the process described in the previous section, and hence it achieves the goal provided that the resources Γ and (9) successfully accomplish their tasks.

However, this is not the best way to represent the assembly world. Although it avoids the representational frame problem, — no need in anything like frame axioms, — the inferential frame problem[6] still remains: in an analysis of the play, after every move, we need to verify that only one $\dot{\wedge}$-conjunct of Γ (which is the major part of the world) changes its $\unrhd$-stage; these changes are not reflected in the current position — Γ remains Γ and we need to separately keep track of in what stages the $\unrhd$-processes of its effect are. It is just the presence of the operators $\rhd$, $\unrhd$, $[\rhd]$, $[\unrhd]$ in resources that makes a trouble of this kind.

A better way to represent the assembly world, which avoids the inferential (along with the representational) frame problem is to view each memory location as an independent resource which can directly accept our commands. Then we can be sure that a command changes only the resource (the contents of the location) to which it is given, and we don't need to check or re-write the other agents of the system as long as their effects don't contain the operators $\rhd$, $\unrhd$, $[\rhd]$, $[\unrhd]$.

Below is a description of this sort of axiomatization for the assembly world. Observe that it is totally $(\rhd, \unrhd, [\rhd], [\unrhd])$-free.

Let

$$\Lambda_1 := \gg\ll\Big(\Big(\dot{\exists}x, y\big(\angle L_2(x)\dot{\wedge}\angle L_3(y)\dot{\wedge}\Uparrow L_1(x+y)\big)\Big)\Lambda_1\Big)$$

$$\Lambda_2 := \gg\ll\Big(\Big(\dot{\exists}x, y\big(\angle L_1(x)\dot{\wedge}\angle L_3(y)\dot{\wedge}\Uparrow L_2(x+y)\big)\Big)\Lambda_2\Big)$$

$$\Lambda_3 := \gg\ll\Big(\Big(\dot{\exists}x, y\big(\angle L_1(x)\dot{\wedge}\angle L_2(y)\dot{\wedge}\Uparrow L_3(x+y)\big)\Big)\Lambda_3\Big)$$

We assume the following axioms:

$$\Big(\Updownarrow L_1(2)\Big)\Lambda_1 \tag{10}$$

[6]For a discussion of these 2 sorts of frame problem, see [12].

$$\left(\Updownarrow L_2(0)\right)\Lambda_2 \tag{11}$$

$$\left(\Updownarrow L_3(0)\right)\Lambda_3 \tag{12}$$

together with $\left((6)\ddot{\curlywedge}(7)\right) \gg$.

Thus, each of the agents (10), (11), (12) accepts one single command, the execution of which results in writing in the corresponding location the sum of the contents of the other two locations. A strategy for achieving the goal $\left(\dot{\top}\rhd \Updownarrow L2(10)\right) \gg$ is: Give a command to (11), then to (12), then to (10) and then, again, to (11). A reasonable algorithm which finds this kind of strategy and verifies its successfulness, would only keep track of the changes that occur in the effect of the resource to which a command is given. As we noted, however, this relaxed behavior of the algorithm is possible only if those effects don't contain the "trouble maker" operators $\rhd$, $\unrhd$, $[\rhd]$ and $[\unrhd]$.

References

[1] W.Bibel, *A deductive solution for plan generation.* New Generation Computing 4 (1986), pp.115-132, 1986.

[2] A.Blass, *A game semantics for linear logic.* Annals of Pure and Applied Logic, v.56 (1992), pp. 183-220.

[3] S.Brüning, S.Hölldobler, J.Shneeberger, U.C.Sigmund and M.Thieshler, *Disjunction in resource-oriented deductive planning.* Technical Report AIDA-94-03, GF Intellectic, FB Informatic, TH Darmstadt, 1994.

[4] K.Dozen and P.Schroeder-Heister, *Substructural Logics.* Studies in Logic and Computation, D.Gabbay (ed.), Clarendon Press, Oxford, 1993.

[5] J.Y.Girard, *Linear logic.* Theoretical Computer Science, v.50-1 (1987), pp. 1-102.

[6] G.Grosse, S.Hölldobler and J.Shneeberger, *Linear deductive planning.* Technical Report AIDA-92-08, GF Intellectic, FB Informatic, TH Darmstadt, 1992.

[7] S.Hölldobler and J.Shneeberger, *A new deductive approach to planning.* New Generation Computing 8(3), pp.225-244, 1990.

[8] G.Japaridze, *A constructive game semantics for the language of linear logic.* Annals of Pure and Applied Logic 85 (1997), no.2, pp.87-156.

[9] G.Japaridze, *A Formalism for Resource-Oriented Planning.* IRCS Report 98-01.

[10] G.Japaridze, *The Logic of Resources and Tasks.* PhD Thesis. University of Pennsylvania, Philadelphia, 1998, 145 pages.

[11] S.C.Kleene, *Introduction to Metamathematics.* New York, 1952.

[12] S.Russell and P.Norwig, *Artificial Intelligence: a Modern Approach.* Prentice-Hall, 1995.

[13] U.Sigmund and M.Thielscher, *Equational Logic Programming, Actions and Change.* Proc. Joint International Conference and Symposium on Logic Programming JICSLP'92, 1992.