On game semantics of the affine and intuitionistic logics (Extended abstract)

Ilya Mezhirov[1] and Nikolay Vereshchagin[2] *

[1] The German Research Center for Artificial Intelligence,
TU Kaiserslautern,
`ilya.mezhirov@dfki.uni-kl.de`
[2] Lomonosov Moscow State University,
Leninskie Gory, Moscow 119991,
`ver@mccme.ru`

Abstract. We show that the uniform validity is equivalent to the non-uniform validity for both Blass' semantics of [1] and Japaridze's semantics of [5] (thus proving a conjecture from [5]). We present a shorter proof (than that of [10]) of the completeness of the positive fragment of intuitionistic logic for both these semantics. Finally, we show that validity for the "parallel recurrence" version of Japaridze's semantics of [5] is equivalent to accomplishability in the sense of [4].

1 Logic of tasks and intuitionistic logic

The linear and affine logics of Girard [3] are often understood as logics of resources. Propositional variables mean certain types of abstract resources (rather than assertions, as in classical logic). Each occurrence of a variable p identifies one unit of a resource p. The connectives $\wedge$ (the multiplicative AND), $\vee$ (the multiplicative OR), $\sqcap$ (the additive AND), $\sqcup$ (the additive OR), $\neg$ (the negation), $!, ?$ (exponential AND and OR) and $\mathbf{0}, \mathbf{1}$ (constants) have the following meaning. The expression $x \wedge y$ means one unit of x and one unit of y (for example, $x \wedge x$ is two units of the resource x). The expression $x \sqcap y$ means an obligation to provide one unit of the resource x or one unit of the resource y where the consumer of resources (the user) makes the choice between x and y (for example, $x \sqcap x$ is the same as x). The formula $x \sqcup y$ means also an obligation to provide one unit of the resource x or one unit of the resource y. However, this time the choice between x and y is made by the provider (again $x \sqcup x$ is the same as x).

What is the interpretation of $\vee$ (multiplicative OR)? Let us use the following metaphor. Assume that resources are coins of different types. Each occurrence of a variable x is regarded as a coin of type x. The coins can be genuine or fake, and the consumer cannot distinguish between genuine and fake ones. The expression $x \vee y$ is understood as a pair of coins, a coin of type x and a coin of

* The work was supported in part by the RFBR grants 06-01-00122

type y, such that at least one of them is genuine (however, the user does not know which one).

The expression $\neg x$ means the obligation of the user to pass to the provider one coin of type x (we can understand x also in this way, as the obligation of the provider to pass to the user one coin of type x). The formula $!\,x$ is understood as an infinite stock of genuine coins of type x. The expression $?\,x$ means an infinite stock of coins of type x such that at least one coin in the stock is genuine (and the user does not know which one). In other words $!\,x$ is a countable version of the multiplicative AND $x \wedge x \wedge x \ldots$, and $?\,x$ is a countable version of the multiplicative OR $x \vee x \vee x \ldots$.

The constants $\mathbf{0}, \mathbf{1}$ are understood as non-accomplishable obligations (tasks): $\mathbf{0}$ is the obligation of the provider and $\mathbf{1}$ is the obligation of the user.

In the next section we give a formal definition of the semantics which clarifies this intuition. From both the technical and philosophical viewpoints, this semantics coincides with Japaridze's "The logic of tasks" semantics defined in [4] and later extended to what has been termed *abstract resource semantics* in [8]. What we call "a coin" is called "a task" in [4]. A genuine coin is a task accomplished by the provider. A false coin is a task that the provider failed to accomplish. We are using a different language, as we think our language is more convenient to present the definition.

1.1 The semantics

Fix a countable set of *variables* $x_1, x_2, \ldots$. A *literal* is a variable x_i or a negated variable $\neg x_i$, called *dual* to x_i. The literals of the form x_i are called *positive* and literals of the form $\neg x_i$ are called *negative.* Formulas are obtained from literals, constants $\mathbf{1}, \mathbf{0}$ and connectives $\wedge, \vee, \sqcap, \sqcup, ?, !$ in the usual way. We consider formulas where negations appear only before variables. When we write $\neg A$, we always mean the formula dual to A, that is, the formula which is obtained from A by changing each connective, each variable and its constant to its dual: $\vee \leftrightarrow \wedge$, $\sqcup \leftrightarrow \sqcap$, $! \leftrightarrow ?$, $\neg x_i \leftrightarrow x_i$ and $\mathbf{1} \leftrightarrow \mathbf{0}$.

Each formula is assigned a two player game $[A]$ of perfect information between *Provider* (or, *Producer*) and *User* (or *Consumer*). If A is derivable in the affine logic then the Consumer will have a winning strategy in the game $[A]$.

First we replace in A each occurrence of a formula of the type $!\,B$ by the formula $B \wedge B \wedge B \ldots$, and each occurrence of a formula of the type $?\,B$ by the infinite formula $B \vee B \vee B \ldots$. It is easy to see that the order of replacements does not affect the result, which is an infinite formula of a finite depth.

In the game $[A]$, the players make moves in turn. It does not matter who moves first. In his turn, Producer may perform any finite sequence $p_1, \ldots, p_k$ of *actions* (the sequence may be empty). Each action p_i has the form "choose the left formula in $B \sqcap C$" or "choose the right formula in $B \sqcap C$", where $B \sqcap C$ is an occurrence of a formula in A.

For each occurrence of a formula $B \sqcap C$ in A Producer may only once make an action of the type "choose something in $B \sqcap C$" and he is unable to change the choices he has made.

In her turn, Consumer may also perform any finite (possibly empty) sequence of *actions*. Each her action has one of the following two forms. (1) "Choose the left (right) formula in $B \sqcup C$" where $B \sqcup C$ is an occurrence of a formula in A. (2) "Allocate U to V", where U is an occurrence of a negative literal in A and V is an occurrence of a positive literal similar to U (the literals are called similar if they have the same variable: x_i and $\neg x_i$ are similar). The informal meaning of this action is that Consumer wants to accomplish her obligation V using the coin U. It is instructive to visualise this action as follows. Imagine that each occurrence of a negative literal $\neg x$ is a box belonging to Producer and containing a coin of type x. Regard all occurrences of positive literals x as empty boxes of type x belonging to Consumer. The action $U \mapsto V$ moves the coin from the box U to the box V. According to this interpretation, we will sometimes call occurrences of the negative literals *Producer's boxes* and occurrences of the positive literals *Consumer's boxes*.

For each occurrence of a positive literal V Consumer may perform at most one action of the form $U \mapsto V$, and for each occurrence of a negative literal U she may perform at most one action of the form $U \mapsto V$ (every box can contain at most one coin). Consumer is not allowed to move coins in the other direction (from her boxes to Producer's ones). Neither is she allowed to move coins from boxes to "nowhere". In other words, actions of type (2) establish a matching between Producer's and Consumer's boxes that respects variable's names. Note that Producer is not allowed move coins at all.

For each occurrence of a formula $B \sqcup C$ in A Consumer may only once make an action of the type "choose something in $B \sqcup C$".

Each play consists of infinite (countably many) number of moves. When the play is finished, we define who has won as follows. Call a *coin evaluation* a mapping e from Producer's boxes to the set $\{\mathtt{f}, \mathtt{g}\}$. If $e(U) = \mathtt{f}$ we say that the coin in the box U is fake, otherwise (if $e(U) = \mathtt{g}$) we say that it is genuine.

Given a coin evaluation, we recursively define for each occurrence of a formula B in A who has won B, Consumer or Producer.

(1) If V is an occurrence of a positive literal, then Consumer has won V iff, in the course of the play, Consumer has performed an action "allocate U to V" and $e(U) = \mathtt{g}$. In other words, if in the end of the game the box V contains a genuine coin. If the box V is empty or contains a fake coin then Consumer has lost V.

(2) If U is an occurrence of a negative literal, then Consumer has won U iff $e(U) = \mathtt{f}$.

Rules (1) and (2) imply the following. If Consumer has performed an action "allocate U to V" then exactly one of U, V is won by Consumer.

(3) Consumer has won an occurrence of a formula $B \wedge C$ iff she has won both B and C. She has won an occurrence of a formula $B \vee C$ iff she has won B or C.

(4) In a similar way we define who has won occurrences of $!B$ and $?B$: an occurrence $B_1 \wedge B_2 \wedge B_3 \ldots$ (we use subscripts to distinguish between different occurrences of B) is won by Consumer iff she has won all the occurrences

$B_1, B_2, B_3 \ldots$. An occurrence $B_1 \vee B_2 \vee B_3 \ldots$ is won by Consumer iff she has won at least one of the occurrences $B_1, B_2, B_3 \ldots$.

(5) Consumer has won an occurrence of $B \sqcup C$ iff, in the course of the play, she has decided between B and C in $B \sqcup C$ and has won the chosen formula. That is, she has performed the action "choose the left formula $B \sqcup C$" and she has won B, or she has performed the action "choose the right formula in $B \sqcup C$" and she has won C. If she has not decided between B and C in $B \sqcup C$ then she has lost $B \sqcup C$.

(6) For a subformula of the form $B \sqcap C$ the definition is similar (we swap Consumer and Producer). Producer has won an occurrence of $A \sqcap B$ iff in the course of the play, he has decided between B and C in $B \sqcup C$ and has won the chosen formula.

(7) Every occurrence of $\mathbf{0}$ is won by Producer and every occurrence of $\mathbf{1}$ is won by Consumer.

Finally, we say that the Consumer has won the play iff for every coin evaluation e she has won the entire formula A.

There is an equivalent way to define who was won the play. In the end of the play replace by $\mathbf{0}$ all the occurrences of the formulas of type $B \sqcup C$ in A such that Consumer has not chosen B or C. Replace by $\mathbf{1}$ all the occurrences of $B \sqcap C$ where Producer has not decided between B and C. Replace each remaining occurrence of a formula of the form $B \sqcup C$ or $B \sqcap C$ by the chosen subformula. (The order of replacements does not affect the result.) Then replace each occurrence of a variable by a new variable in such a way that matching occurrences are replaced by the same variables and non-matching occurrences by different ones. (We call *matching* the occurrences of x in U and V such that Consumer has allocated U to V.) The resulting formula is an infinite formula of finite depth with connectives $\vee, \wedge$ and constants $\mathbf{1}, \mathbf{0}$. Consumer has won the play iff this formula is a classical tautology (where $\wedge$ is understood as AND, $\vee$ as OR, and $\neg x$ as the negation of x).

The above described transformation is what is called *elemntarization* in [4–6]

The definition of the game $[A]$ is completed. We call a formula A *accomplishable*[3] if Consumer has a winning strategy in the game $[A]$. We call a formula A *computably accomplishable* if Consumer has a computable winning strategy in the game $[A]$.

If A has no exponential connectives then the game $[A]$ is essentially finite (every player can perform only finitely many actions) and thus the game $[A]$ is accomplishable iff it is computably accomplishable. In this case at least one of the players has a (computable) winning strategy.

In the general case it is unknown whether accomplishability is equivalent to computable accomplishability.

Note that the rules of the game favour Producer. For example, it might happen that Producer has a winning strategy in both games A and $\neg A$. This happens, say for $A = x$.

[3] We use here the terminology of [4]

The simplest accomplishable formula is $x \vee \neg x$: in order to win Consumer just moves Producer's coin to her box.

Important remark. It is important that Consumer cannot distinguish fake and genuine coins. (Formally, that means that we choose coin evaluations after the play is finished.) That is why the formula $A = (\neg x \wedge \neg x) \vee x$, which is not derivable in the affine logic, is not accomplishable. In the game $[A]$ Consumer has essentially two strategies: (1) she moves the first of Producer's coins (2) she moves the second one. Both strategies do not win. Indeed, if only one coin is genuine, namely, the coin that has not been moved, Consumer looses the formula A.

The definition of an accomplishable formula is very robust: we can change the definition of the game in many ways so that the class of (computably) accomplishable formulas does not change. Below we present several such modifications.

1. We can forbid Consumer (or Producer, or both) to perform several actions in one move. Indeed, postponing actions never hurts the player. By the same reason it does not matter who starts the play.[4]

2. We can ban all actions inside occurrences B and C belonging to an occurrence of $B \sqcup C$ [or $B \sqcap C$] such the choice action has not yet been applied to $B \sqcup C$ [$B \sqcap C$, respectively].

3. We can assume that all the Producers boxes are empty at the start of the play and he is allowed to perform an action "deposit a coin in a box". Consumer is allowed to apply action "allocate U to V" only when U is not empty. At the end of a play a Producer's box is won by him iff he has deposited a genuine coin in it.

4. We can define the game $[A]$ recursively. To this end we need a notion of a "game with coins" and operations on such games that correspond to connectives.

Games of the form $[A]$ can be regarded as vending machines: Consumer's boxes can be identified with slots for coins and Producer's boxes with compartments for releasing the products. For instance, a vending machine that accepts 1 Euro coin and 50 cents coins and sells coffee and tee, for 1 Euro each, can be represented by the formula

$$!(\neg(\text{1 Euro} \sqcup (\text{50 cents} \wedge \text{50 cents})) \vee (\text{tee} \sqcap \text{coffee})).$$

Informally, the game $[A]$ is accomplishable iff the vending machine can work without having any resources in advance.

Historical remarks. As we have said, the notion of an accomplishable formula was defined in [4] (Logic of Tasks semantics). That paper also provides a sound and complete calculus for additive fragment of accomplishable formulas. For the multiplicative fragment an equivalent semantics was defined in [8] (Abstract Resource Semantics) together with a sound a complete calculus (CL5). Finally, the calculus CL4 from [8] provides a sound and complete axiomatisation for the set of all accomplishable formulas that have no exponential connectives. It is

[4] The games having that property are called static, see e.g. [5]. We will consider other static games in the next section.

unknown whether the entire set of accomplishable formulas is computably enumerable. The similar question is open for computably accomplishable formulas as well.

1.2 Accomplishable formulas and affine logic

We will use the following variant of affine logic. A *sequent* is a finite list of formulas. A sequent consisting of formulas $A_1, \ldots, A_n$ is denoted by $\vdash A_1, \ldots, A_n$. The order of formulas in a sequent does not matter, but the multiplicity of each formula matters. That is, $\vdash p, p$ and $\vdash p$ are different sequents.

Axioms: $\vdash A, \neg A, \quad \vdash \mathbf{1}$.

Derivation rules:

$\frac{\vdash \Gamma, A \quad \vdash \Delta, \neg A}{\vdash \Gamma, \Delta}$ (Cut). $\quad \frac{\vdash \Gamma, A \quad \vdash \Delta, B}{\vdash \Gamma, \Delta, (A \wedge B)}$ (Introducing $\wedge$).

$\frac{\vdash \Gamma, A, B}{\vdash \Gamma, (A \vee B)}$ (Introducing $\vee$). $\quad \frac{\vdash \Gamma, A \quad \vdash \Gamma, B}{\vdash \Gamma, (A \sqcap B)}$ (Introducing $\sqcap$).

$\frac{\vdash \Gamma, A}{\vdash \Gamma, (A \sqcup B)}$, $\quad \frac{\vdash \Gamma, B}{\vdash \Gamma, (A \sqcup B)}$ (Introducing $\sqcup$). $\quad \frac{\vdash \Gamma}{\vdash \Gamma, A}$ (Weakening).

$\frac{\vdash \Gamma, A}{\vdash \Gamma, ?\, A}$ (Dereliction). $\quad \frac{\vdash \Gamma, ?\, A, ?\, A}{\vdash \Gamma, ?\, A}$ (Contraction). $\quad \frac{\vdash ?\, \Gamma, A}{\vdash ?\, \Gamma, !\, A}$ (R).

Here $?\, \Gamma$ denotes the list obtained from Γ by prefixing by ? all formulas in the list.

Call the formula $A_1 \vee \ldots \vee A_n$ the *formula image* of the sequent $\vdash A_1, \ldots, A_n$. If the list is empty (that is, $n = 0$), its formula image is equal to $\mathbf{0}$. A sequent is called *(computably) accomplishable* if so is its formula image.

Theorem 1. *The set of accomplishable sequents contains all axioms of the affine logic and is closed under all its derivation rules and under the substitution. The same applies to computable accomplishable formulas. (Hence all derivable formulas are computably accomplishable.)*

The affine logic is not complete with respect to accomplishability semantics. An example of an accomplishable non-provable formula is

$$[(\neg a \vee \neg b) \wedge (\neg c \vee \neg d)] \vee [(a \vee c) \wedge (b \vee d)].$$

(This formula was used by Blass in [1] to show that the affine logic is incomplete with respect to his semantics.)

1.3 Accomplishable formulas and the intuitionistic propositional calculus IPC

Consider the Girard's translation from the language of propositional formulas with connectives $\wedge, \vee, \rightarrow, \perp$ into the language of affine logic. Each formula A is assigned a formula A^* defined recursively: $(x_i)^* = x_i$ and

$$(A \vee B)^* = !\, A^* \sqcup !\, B^*, \quad (A \wedge B)^* = A^* \sqcap B^*,$$
$$(A \rightarrow B)^* = \neg(!\, A^*) \vee B^* = ?(\neg A^*) \vee B^*, \quad \perp^* = \mathbf{0}.$$

This translation preserves provability. The next lemma shows that the combination of this translation with accomplishability semantics yields a sound semantics for the intuitionistic calculus.

Lemma 1. *The set L of all formulas A whose translation A^* is accomplishable is a super-intuitionistic logic (that is, L contains all axioms of IPC and is closed under Modus Ponens and substitution). The same holds for the set of all formulas whose translation is computable accomplishable.*

It turns out that this semantics of IPC is complete for the positive fragment of IPC.

Theorem 2. *If A is a positive formula (that is, it does not contain $\bot$) and A^* is accomplishable then A is derivable in IPC.*

One of the technical tools in the proof of this theorem is the following theorem, which was stated by Medvedev in [11] (without a complete proof) and proved in [2]. A *critical implication* is a (positive) formula of intuitionistic language of the form $A_1 \wedge \dots \wedge A_m \to R$, where R is an OR of variables and each A_i has the form $(P_i \to Q_i) \to Q_i$. Here every P_i is an AND of variables and Q_i is an OR of variables and for all i the formulas P_i and Q_i have disjoint sets of variables. The number of variables in R, P_i and Q_i is positive and may be equal to 1.

Theorem 3 ([11, 2]). *If a super-intuitionistic logic contains no critical implication then its positive fragment coincides with the positive fragment of IPC. Moreover, if a positive formula A is not derivable in IPC then there is a critical implication J such that IPC $\vdash (A' \to J)$ where A' is a formula obtained from A by a substituting certain formulas of type OR-of-AND-of-variables for A's variables.*

Theorem 2 does not generalise to negative formulas. We know two examples of a formula A such that A^* is accomplishable but A is not provable in IPC: Rose formula from [12] and Japaridze's formula from [7]. The latter one is simple enough to present it here:

$$(\neg p \to x \vee y) \wedge (\neg\neg p \to x \vee y) \to (\neg p \to x) \vee (\neg p \to y) \vee (\neg\neg p \to x) \vee (\neg\neg p \to y). \tag{1}$$

Here $\neg B$ is an abbreviation for $B \to \bot$.

Note that if we defined the translation of $A \vee B$ as just $A^* \sqcup B^*$ then the translation of the axiom

$$(x \to r) \to ((y \to r) \to (x \vee y \to r))$$

would not be accomplishable.

2 Japaridze's game semantics and accomplishability

2.1 Static games

We will use a rather general notion of games from [5] between two players, called Environment and Machine.

A *move* is a string over the keyboard alphabet. A *labelled move (labmove)* is a move prefixed by `E` or `M` (the prefix indicates who has done the move, Environment or Machine). A *run* is a finite or infinite sequence of labmoves. A *position* is a finite run.

A *game*[5] is specified by a set of runs L and a function W mapping runs to the set $\{\mathtt{E},\mathtt{M}\}$. All runs in L are called *legal*, all other runs are called *illegal.* If $W(\Gamma) = P$, we say that the run Γ is *won* by P. Otherwise it is *lost* by P.

The set L must have the following properties: (1) the empty sequence (called the *initial position*) belongs to L and (2) if a (finite or infinite) run Γ is in L then all its finite prefixes are in L too.

Let $\Gamma = \alpha_1, \alpha_2, \dots$ be a run and α_i a labmove in that run. We say that α_i is the first *first illegal* labmove in Γ if $\alpha_1, \dots, \alpha_{i-1}$ is legal run but $\alpha_1, \dots, \alpha_i$ is not. Each illegal run Γ has exactly one first illegal labmove. The function W must have the following property: every illegal run is lost by that Player who has made the first illegal move in it.

We have not yet defined how to play a game. It is not obvious since in some positions both players can make a legal move and the rules do not specify who has the turn to play in such a position.

There are eight ways to play a game. We have to make three choices: who starts the game, how many moves (at most one or several) is Environment allowed to make in its turn, and how many moves (at most one or several) is Machine allowed to make in its turn. For example, the game can be played as follows: Environment starts the play; in its turn, each player either makes a move or passes. Another way: Machine starts the play; in its turn, Environment can make any finite sequence of moves (including the empty sequence); in its turn, Machine can make a move or pass. For all ways, we assume that the game lasts infinitely long and the turn to play alternates.

For certain games it is crucial which of the eight modes to play is chosen (for example, if $W(\Pi) = \mathtt{E}$ if Π starts with a move of `E` and $W(\Pi) = \mathtt{M}$ otherwise). There are however two important classes of games, *strict* games and more general *static* games, for which it does not matter.

A game is called *strict* if for every legal position Δ at most one player can play a move α so that the resulting position Δ, α is legal.

Most games considered in the literature are strict ones. However, the operation on games we are going to define do not look natural when applied to strict games. They look natural when applied to another type of games (called static games) defined in the next two paragraphs. Informally, static games are those games in which it never hurts the player to postpone moves.

Let Γ, Δ be (finite or infinite) runs. We say that Δ is a *Machine-delay* of Γ if Δ is obtained from Γ by postponing certain Machine's moves (may be infinitely many). Formally, the following conditions should hold. (1) Erasing all moves of Environment in Γ and Δ results in the same run; the same holds for erasing Machine's moves. (2) For all k and l if kth move of Machine is made later than lth move of Environment in Γ then so is in Δ.

[5] called a *constant game* in [5]

We define a notion of *Environment-delay* in a similar way. We say that the game is *static* if the following holds for every player P, every run Γ and every P-delay Δ of Γ. (1) If P has not made the first illegal move in Γ then P has not made the first illegal move in Δ either. (2) If Γ is won by P then so is Δ.

We call a static game *winnable* if Machine has a winning strategy in the game. It does not matter which of the above eight ways to play the game to choose: the class of winnable games is robust under switching between the eight playing modes. However, it is important that we do not force any player to make a move in its turn. We call a static game *computably winnable* if Machine has a computable winning strategy in the game (that is, there is a Turing machine that wins the game).

Now we will define operations $\neg, \wedge, \vee, \sqcup, \sqcap, ?, !$ on games. Those operation will preserve static property. We will then call a formula of affine language (computably) winnable if every substitution of static games for variables results in a (computably) winnable game. One of our main result states that a formula is (computably) winnable iff it is (computably) accomplishable.

2.2 Operations on games

The operation of *negation* $\neg$ just swaps the roles of players: Machine plays in Environment's role and vice versa. The set of legal runs of $\neg A$ is obtained from that of A by replacing each run by its dual (a run Γ' is dual to Γ if it is obtained from Γ by exchanging labels `E` and `M` in all labmoves). Machine wins a run Γ in $\neg A$ iff the dual run Γ' is won by Environment in A.

The *choice conjunction* applied to games A, B produces the following game $A \sqcap B$. Environment decides between A and B. Then the chosen game is played. If Environment has not decided, it looses. Formally a non-empty run is legal iff it starts with Environment's move "choose left" of "choose right" and the rest of the run is the legal run of A if the first move is "choose left" and is the legal run of B if the first move is "choose right". A legal run is won by Machine in the following three cases: (1) it is empty, (2) the first move is "choose left" and the rest of the run is won by Machine in the game A, and (3) the first move is "choose right" and the rest of the run is won by Machine in the game B.

The *choice disjunction* $A \sqcup B$ of A, B is dual to $A \sqcap B$. This time Machine has to decide between A and B (and it looses if it has not decided). In other words, $A \sqcup B = \neg(\neg A \sqcap \neg B)$.

Parallel disjunction $\vee$. In the game $A \vee B$ the players play two games A and B simultaneously. In order to win, Machine has to win at least one game. Formally, a run Γ is legal if the following holds. Let Γ^i denote the result of removing from Γ all the labmoves that do not have the form $Pi.\alpha$ (where $P = \mathtt{E}, \mathtt{M}$) and replacing the prefix $Pi.\alpha$ by $P\alpha$ in all the remaining labmoves. A run Γ is legal if every its labmove has the form $Pi.\alpha$ (where $i = 1, 2$) and the runs Γ^1, Γ^2 are legal runs of A, B, respectively. Such a run is is won by Machine if either Γ^1 is won by Machine in A or Γ^2 is won by Machine in B (or both).

Parallel conjunction $\wedge$ is dual to parallel disjunction. The difference is that this time Machine has to win both games. In other words, $A \wedge B = \neg(\neg A \vee \neg B)$.

Parallel recurrence $\curlywedge$. The game $\curlywedge A$ is essentially an infinite parallel conjunction $A \wedge A \wedge A \wedge \ldots$. Players play simultaneously infinite plays and in order to win Machine has to win all plays. Formally, a run is legal if all its moves have the form $Pi.\alpha$ where $i = 1, 2, 3, \ldots$ (we spell natural numbers in decimal notation, say) and all $\Gamma^1, \Gamma^2, \Gamma^3, \ldots$ are legal runs of A. Such a run is won by Machine if all Γ_i are won by Machine in the game A.

Parallel corecurrence $\curlyvee$ is dual to parallel recurrence $\curlywedge$. Again players play infinitely many plays in the game A. However, this time in order to win, Machine has to win at least one play. That is, $\curlyvee A = \neg(\curlywedge \neg A)$.

All these operations preserve the static property. However not all of them preserve strictness property. Consider, for example, the multiplicative conjunction. Assume that the games A and B are strict, the first move in A is made by Environment and the first move in B is made by Machine. Then the first move in $A \wedge B$ can be made by either player.

Nevertheless all eight ways to play $A \wedge B$ are equivalent (for all strict games A, B). So we could fix any of the eight ways to play and thus convert $A \wedge B$ into an equivalent strict game. However such stipulating would be quite unnatural (bureaucratic, using Japaridze's word). As a result, we would not have the property $A \wedge B = \neg(\neg A \vee \neg B)$. The games $A \wedge B$ and $\neg(\neg A \vee \neg B)$ would be only equivalent in a sense.

Another reason to prefer static games is that all the computational problems can be quite naturally expressed as static games and not as strict games (see [6] for many examples). That is why Japaridze has chosen static games as a basis for his Game semantics.

2.3 Winnable = accomplishable

Let $A(x_1, \ldots, x_n)$ be a formula of affine language and $G_1, \ldots, G_n$ static games. Substituting G_i for x_i in A and performing all operations spelled in A we obtain a static game $A(G_1, \ldots, G_n)$. Exponential AND and OR are interpreted as $\curlywedge, \curlyvee$ respectively. We say that a formula A is *winnable*, if for all static games $G_1, \ldots, G_n$ the resulting game $A(G_1, \ldots, G_n)$ is winnable. We say that A is *computably winnable*, if for all static games $G_1, \ldots, G_n$ the game $A(G_1, \ldots, G_n)$ is computably winnable.

Consider also the uniform version of winnability. Call a formula A is *uniformly* (computably) winnable, if there is a (computable) strategy winning the game $A(G_1, \ldots, G_n)$ for all static games $G_1, \ldots, G_n$. We will see that winnability coincides with accomplishability and computable winnability coincides with computable accomplishability.

Note that in the definition of winnability we do not require the games $G_1, \ldots, G_n$ be determined.[6] Why? The class of determined games is closed under all operations considered. And who wins the game is determined just classically: Machine wins (i.e., it has a winning strategy in) $A \wedge B$ iff it wins A and wins B, Machine

[6] The game is called *determined* in either Machine, or Environment has a winning strategy in the game.

wins $A \vee B$ if it wins A or wins B etc. Thus if we restrict the class of static games by determined ones, a formula would be winnable iff it is a classical tautology.

For uniform winnability and computable winnability this is not the case: for example, it might be that Machine has a computable winning strategy in the game $A \vee B$ but does not have computable winning strategy in either A or B. Therefore for these versions of winnability it seems quite natural to restrict the class of games to determined ones. However, it turns out that this restriction does not affect the classes of uniformly winnable, computably winnable and uniformly computably winnable formulas.

Theorem 4. *The following three properties of a formula A are equivalent:*
(1) A is computably accomplishable,
(2) A is uniformly computably winnable, and
(3) For every 2-moves[7] *(hence determined) static games $G_1, \dots, G_n$ the game $A(G_1, \dots, G_n)$ is computably winnable.*

The equivalence of (2) and (3) implies that every winnable formula is uniformly computably winnable. This proves Conjecture 26.1 from [5] (a similar statement for branching recurrences follows from Theorem 6 below).

Theorem 5. *The following four properties of a formula A are equivalent:*
(1) A is accomplishable.
(2) A is uniformly winnable.
(3) A is winnable.
(4) There is a Machine's strategy winning every game of the form $A(G_1, \dots, G_n)$, where $G_1, \dots, G_n$ are 2-moves static games.

These theorems together with Theorem 2 and Lemma 1 show that winnability is a sound complete game theoretic semantics for the intuitionistic propositional calculus.

Corollary 1. *A positive formula is provable in IPC if and only if the formula A^* is computably winnable. The same holds for winnability, uniform winnability and uniform computable winnability.*

2.4 Countable branching recurrence

There is another way to interpret exponential connectives as operations on games, called *branching recurrence* and *corecurrence* in [5]. There are two versions of them, countable and uncountable. Countable branching recurrence and corecurrence were essentially introduced by Blass in [1].

The countable branching recurrence $⫰^{\aleph_0}$ is the operation on games defined as follows. In the game $⫰^{\aleph_0} A$ players play countably many plays in game A and Machine has to win all plays (like in the game $\curlywedge A$). However this time its job is even more difficult, as Environment may “copy” positions. Informally, we can

[7] A game is a k-move game if every legal run has at most k labmoves.

imagine that a position in the game $⫰^{\aleph_0}A$ is a tuple $\langle p_1, \ldots, p_n\rangle$ of positions of A. The initial position is the tuple $\langle p_1\rangle$ where p_1 is the initial position in A. If the current position is $\langle p_1, \ldots, p_n\rangle$, then each player is allowed to make a legal move in any of the positions $p_1, \ldots, p_n$. Environment is also allowed to copy any p_i, in which case the new position is equal to $\langle p_1, \ldots, p_n, p_i\rangle$.

This definition is very informal because we have defined moves in terms of positions and not the other way around, as our framework prescribes.

Moreover, the decsribed game may be not static.[8] To obtain an equivalent static game we need to understand a copying operation as splitting one position (the parent) into two new positions (the children). If a move made in a position P is postponed so that at the time when it is made the position P has been splitted, then that move is automatically played in all descendants of P. More specifically, we assign to each position an address, which is a binary string rather than a natural number. When a position with address w is splitted, the children positions receive addresses $w0$ and $w1$. When a play is finished we obtain a finite or infinite tree consisting of all addresses used. If that tree is finite then all its leaves are addresses of the played games. If the tree is infinite then some infinite paths (say $010101\ldots$) are not addresses of “real” plays. Only those infinite paths are addresses of real plays which have finite number of 1s.

Formally, a run Γ is legal if it is a sequence of labmoves of the form $Pw.\alpha$ and $\mathtt{E}$(split w) having the following two properties. (1) For every w and every proper prefix u of w every occurrence of a labmove of the form $Pw.\alpha$ or $\mathtt{E}$(split w) is preceeded by exaclty one occurence of the labmove $\mathtt{E}$(split u). (2) For every infinite string w that has only finitely many 1s consider the sequence $\Gamma(w)$ of all labmoves in Γ of the form $Pu.\alpha$—with “$u.$” removed—where u is a prefix of w. For each w that has only finitely many 1s the run $\Gamma(w)$ must be a legal run of A. A legal run Γ is won by Machine if $\Gamma(w)$ is won by Machine for all w that has only finitely many 1s.

In the definition of *uncountable branching recurrence* $⫰$, we stipulate that Γ is won be Machine if $\Gamma(w)$ is won for all infinite w (and not only for w having finite number of 1s). Thus the difference between countable and uncountable versions is due to different understandings which plays are “real” and which are not.

Countable branching corecurrence $⫯^{\aleph_0}$ is defined in the dual way so that $⫯^{\aleph_0}A = \neg(⫰^{\aleph_0}\neg A)$.

Let us change the interpretation of ! and ? to $⫰^{\aleph_0}$ and $⫯^{\aleph_0}$, respectively. We obtain new notions of winnability and computable winnability, which does not coincide with the old ones (and hence differ from accomplishability). Indeed, it is not hard to see that the formula

$$?(\neg x \sqcap \neg y) \vee (!\,x \sqcup !\,y),$$

[8] Indeed, assume that the initial position is lost by Machine and every run of length 1 is won by Machine. Then the position $\mathtt{M}1.\alpha, \mathtt{E}$(copy the first position) is won by Machine but the position $\mathtt{E}$(copy the first position)$, \mathtt{M}1.\alpha$ is lost.

is not accomplishable but is uniformly computably winnable provided exponentials are interpreted as countable branching recurrences.

However, Theorems 4 and 5 remain partially valid for the new notions of winnability.

Theorem 6. *The following two properties of a formula A are equivalent (if exponentials are interpreted as countable branching recurrences):*
(1) A is uniformly computably winnable, and
(2) For every 2-moves (hence determined) static games $G_1, \dots, G_n$ the game $A(G_1, \dots, G_n)$ is computably winnable.

Theorem 7. *The following three properties of a formula A are equivalent:*
(1) A is uniformly winnable.
(2) A is winnable.
(3) There is a Machine's strategy winning every game of the form $A(G_1, \dots, G_n)$, where $G_1, \dots, G_n$ are 2-moves static games.

Corollary 1 remains true for countable branching recurrences and, moreover, in the translation of $A \vee B$ we may omit !. More specifically, let $(x_i)^\dagger = x_i$ and

$$(A \vee B)^\dagger = A^\dagger \sqcup B^\dagger, \quad (A \wedge B)^\dagger = A^\dagger \sqcap B^\dagger,$$
$$(A \to B)^\dagger = \neg(!\, A^\dagger) \vee B^\dagger = ?(\neg A^\dagger) \vee B^\dagger, \quad \bot^\dagger = \mathbf{0}.$$

Theorem 8. *The set of all formulas A such that $A^\dagger$ is winnable is a superintuitionistic logic (if exponentials are interpreted as countable branching recurrences). The same holds for computable winnability. On the other hand, if a positive formula is not provable in IPC then the formula $A^\dagger$ is not winnable and there are 3-moves games such that the game $A^\dagger(G_1, \dots, G_n)$ is not computably winnable.*

Theorem 8 is true for the Girard's translation as well.

2.5 Historical and terminological remarks

The notions of accomplishability and computable accomplishability come back to [4]. Later they were extended to what has been termed *abstract resource semantics* in [8]. The notions of a computably winnable and uniformly computably winnable formula were defined in [5] under the name a *(uniformly) valid* formula. The uncomputable versions of these were also considered in [5], as a property of games rather than a property of formulas. The notion of a winnable formula (only for countable branching recurrence) was first considered by Blass in [1] (although Blass has considered only strict games and his definition of a countable branching recurrence differs in some technical details from the above definition, the class of winnable formulas is the same).

2.6 Uncountable branching recurrence

Theorem 8 is true for uncountable branching recurrence as well, which was shown in [10]). Our proof of Theorem 8 (based on Theorem 3) also works for uncountable branching recurrences and provides a shorter proof than that of [10].

3 Summary of results

We have conidered the following four classes of formulas in the language of affine logic:

(1) accomplishable formulas,
(2) computably accomplishable formulas,
(3) winnable formulas for parallel recurrences,
(4) computably winnable formulas for parallel recurrences,
(5) winnable formulas for countable branching recurrences,
(6) computable winnable formulas for countable branching recurrences.

We have shown that (1)=(3) and (2)=(4). It is unknown whether (1)=(2) and (3)=(4). Both classes (1) and (2) are different from (3) and (4). All the classes (3)–(6) coincide with their uniform versions.

It is unknown whether the classes (1)–(4) are decidable or computably enumerable.

Both winnability and computable winnability (both parallel and branching versions) provide a sound semantics for the intuitionistic propositional calculus, which is complete for its positive fragment.

Acknowledgements

We are sincerely grateful to Giorgi Japaridze for explaining many subtle things about game operations, pointing to the relevant papers and many useful comments on previous versions of the paper. We are grateful to Alexander Shen and other participants of Kolmogorov seminar at Moscow State University for bringing our interest to game semantics of the intuitionistic logic.

References

1. Andreas Blass. A game semantics for linear logic. Annals of Pure and Applied Logic 56 (1992) 183–220.
2. Chernov A. V., Skvortsov D. P., Skvortsova E. Z., Vereshchagin N. K. Variants of Realisability for Propositional Formulas and the Logic of the Weak Law of Excluded Middle. *Proceedings of Computer Science Logic'02*, Lecture Notes in Computer Science, 2002, v. 2471, pp. 74–88.
3. G.Y. Girard. Linear logic. Theoretical Computer Science 50 (1) (1987) 1–102.
4. Giorgi Japaridze. The logic of tasks. Annals of Pure and Applied Logic 117 (2002) 263–295.
5. G. Japaridze, Introduction to computability logic. Annals of Pure and Applied Logic, vol. 123 (2003), p. 1-99.
6. G. Japaridze. In the beginning was game semantics. arXiv:cs.LO/0507045, 2005. 111 pages.
7. Giorgi Japaridze, A letter to N.V. of 04.11.2005.
8. Giorgi Japaridze. Introduction to Cirquent Calculus and Abstract Resource Semantics. Journal of Logic and Computation 2006 16(4):489-532

9. G. Japaridze. Intuitionistic computability logic. Acta Cybernetica 18 (2007), No. 1, pp. 77-113.
10. G. Japaridze. The intuitionistic fragment of computability logic at the propositional level. Annals of Pure and Applied Logic 147 (2007), No.3, pp.187-227.
11. Yu. T. Medvedev. Finite problems. *Doklady Akad. Nauk SSSR*, 3 (1962) 227–230.
12. G. F. Rose. Propositional calculus and realizability. *Transactions of the American Mathematical Society*, v. 75, N. 1, 1953, p. 1–19.