Information and Computation 206 (2008) 1443–1475

Contents lists available at ScienceDirect

Information and Computation

journal homepage: www.elsevier.com/locate/ic

Sequential operators in computability logic

Giorgi Japaridze [1]

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

ARTICLE INFO

Article history:
Received 10 December 2007
Revised 15 October 2008
Available online 5 November 2008

Keywords:
Computability logic
Interactive computation
Game semantics
Linear logic
Constructive logics

ABSTRACT

Computability logic (CL) is a semantical platform and research program for redeveloping logic as a formal theory of computability, as opposed to the formal theory of truth which it has more traditionally been. Formulas in CL stand for (interactive) computational problems, understood as games between a machine and its environment; logical operators represent operations on such entities; and "truth" is understood as existence of an effective solution, i.e., of an algorithmic winning strategy.

The formalism of CL is open-ended, and may undergo series of extensions as the study of the subject advances. The main groups of operators on which CL has been focused so far are the *parallel*, *choice*, *branching*, and *blind* operators, with the logical behaviors of the first three groups resembling those of the multiplicatives, additives and exponentials of linear logic, respectively. The present paper introduces a new important group of operators, called *sequential*. The latter come in the form of sequential conjunction and disjunction, sequential quantifiers, and sequential recurrences ("exponentials"). As the name may suggest, the algorithmic intuitions associated with this group are those of sequential computations, as opposed to the intuitions of parallel computations associated with the parallel group of operations. Specifically, while playing a parallel combination of games means playing all components of the combination simultaneously, playing a sequential combination means playing the components in a sequential fashion, one after one.

The main technical result of the present paper is a sound and complete axiomatization of the propositional fragment of computability logic whose vocabulary, together with negation, includes all three—parallel, choice and sequential—sorts of conjunction and disjunction. An extension of this result to the first-order level is also outlined.

1. Introduction

This article is yet another addition to the evolving list [5–17] of papers devoted to developing *computability logic* (CL). Baptized so in [5], in a broad sense, CL is not a particular syntactic system or a particular semantics for a particular collection of logical operators, but rather a general platform and an ambitious program for redeveloping logic as a formal theory of computability, as opposed to the formal theory of truth which it has more traditionally been. Formulas in CL stand for computational problems, logical operators represent operations on problems, "truth" is understood as existence of an algorithmic solution, and proofs encode such solutions. Among the main goals of CL at the present stage of development is finding axiomatizations for incrementally expressive fragments of it. Considerable advances have already been made in this direction, and the present paper tells one more success story.

E-mail address: giorgi.japaridze@villanova.edu
URL: http://www.csc.villanova.edu/~japaridz/

[1] This material is based upon work supported by the National Science Foundation under Grant No. 0208816.

0890-5401/$ - see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.ic.2008.10.001

The traditional theory of computation has been primarily evolving around batch computation, despite the fact of life that most tasks performed by computers and computer networks (as well as humans in everyday life) are interactive. Aiming at being a comprehensive formal theory of computation, CL understands computational problems and computability in their most general—*interactive*—sense. And interactive problems are formalized as games played by a *machine* (computer, robot) against its *environment* (user, nature), with *computability* meaning existence of a machine that wins the game against any possible (behavior of the) environment.

Technically, the semantics of CL is thus a game semantics. Among the features distinguishing it from other game-semantical approaches, including Blass's approach [2,3] which is the closest precursor of computability logic, one should point out the following.

First of all, in CL, machine's (proponent's, $\exists$-player's) strategies are limited to algorithmic ones. This is a minimal condition that a game semantics should satisfy if it is meant to find applications in computer science. Due to the same condition, CL has good—semantically rather than syntactically justified—claims to be a *constructive logic*.

Second, players' strategies are no longer considered as functions from positions (the sequences of the previously made moves) to moves. Rather, they are defined in terms of interactive machines, where computation is one continuous process interspersed with—and influenced by—multiple "input" (environment's moves) and "output" (machine's moves) events. A good game semantics is or should be about interaction, while functions are inherently non-interactive. The traditional *strategies-as-functions* approach misses this very important point and creates an unnatural hybrid of interactive (games) and non-interactive (functions) entities. To appreciate the difference, it would be sufficient to reflect on the behavior of one's personal computer. The job of your computer is to play one long—potentially infinite—game against you. Now, have you noticed your faithful servant getting slower every time you use it? Probably not. That is because the computer is smart enough to follow a non-functional strategy in this game. If its strategy was a function from positions (interaction histories) to moves, the response time would inevitably keep worsening due to the need to read the entire—continuously lengthening and, in fact, practically infinite—interaction history every time before responding. Defining strategies as functions of only the latest moves (rather than entire interaction histories) in Abramsky and Jagadeesan's [1] tradition is also not a way out, as typically more than just the last move matters. Back to your personal computer, its actions certainly depend on more than your last keystroke. Thus, the difference between the traditional *functional* strategies and the *post-functional* strategies of CL is not just a matter of taste or convenience. It will become especially important when it comes to (yet to be developed) interactive complexity theory: hardly any meaningful interactive complexity theory can be done with the strategies-as-functions approach. And complexity issues will inevitably come forward when computability logic or similar approaches achieve a certain degree of maturity: nowadays, 95% of the theory of computation is about complexity rather than just computability.

Third, the concept of games that CL deals with is more general than the traditional concepts. Among the distinguishing features of CL games is the absence of *procedural rules*—rules regulating which player can or should move in any given position, the most typical procedural rule being the one according to which the players take turns in an alternating order. In CL games, both players may have legal moves in a given situation. It has been repeatedly argued that only this flexible approach allows us to adequately model truly interactive real-life computational tasks and account for phenomena such as asynchronous communication, concurrency and parallelism. So, again, this difference is not just a difference of tastes, and will certainly play a crucial role when it comes to interactive computational complexity and various flavors of it.

Time has not yet matured for seriously addressing complexity issues though, and CL, including the present paper, continues to be focused on just computability, where there still are too many open questions calling for answers.

The formalism of CL is open-ended, and is expected to undergo series of extensions as the study of the subject advances. The main groups of operations studied so far are:

- *Constant elementary games* (0-ary operations): $\top$, $\bot$.
- *Negation*: $\neg$.
- *Choice operations*: $\sqcap$ (conjunction), $\sqcup$ (disjunction), $\bigsqcap$ (universal quantifier), $\bigsqcup$ (existential quantifier).
- *Parallel operations*: $\wedge$ (conjunction), $\vee$ (disjunction), $\bigwedge$ (universal quantifier), $\bigvee$ (existential quantifier), $\curlywedge$ (recurrence), $\curlyvee$ (corecurrence).
- *Blind operations*: $\forall$ (universal quantifier), $\exists$ (existential quantifier).
- *Branching operations*: $\mathbin{!}$ (recurrence), $\mathbin{?}$ (corecurrence) and a series of their restricted versions such as $\mathbin{!}^{\aleph_0}$ (countable recurrence), $\mathbin{?}^{\aleph_0}$ (countable corecurrence).

There are also various *reduction* operations: $\rightarrow$, defined by $A \rightarrow B = \neg A \vee B$; $\succ\!\!-$, defined by $A \succ\!\!- B = \curlywedge A \rightarrow B$; $\circ\!\!-$, defined by $A \circ\!\!- B = \mathbin{!} A \rightarrow B$; etc.

The present paper introduces the following new group:

- *Sequential operations*: $\triangle$ (conjunction), $\triangledown$ (disjunction), $\bigtriangleup$ (universal quantifier), $\bigtriangledown$ (existential quantifier), $\mathbin{\triangle}$ (recurrence), $\mathbin{\triangledown}$ (corecurrence),

which also induces the reduction operation $\rightarrowtriangle$ defined by $A \rightarrowtriangle B = \mathbin{\triangle} A \rightarrow B$.

The main technical result of this paper is constructing a sound and complete axiomatization for the propositional fragment of CL whose logical vocabulary consists of $\top$, $\bot$, $\neg$, $\wedge$, $\vee$, $\sqcap$, $\sqcup$, $\triangle$, $\triangledown$. An extension of this result to the first-order level that additionally includes the quantifiers $\bigsqcap$,$\bigsqcup$,$\forall$,$\exists$ is also outlined.

2. A tour of the zoo

In this section, we give a very brief and informal overview of the language of computability logic and the game-semantical meanings of its main operators for those unfamiliar with the subject. In what follows, $\top$ and $\bot$ are symbolic names for the players to which we referred as the machine and the environment, respectively.

First of all, it should be noted that computability logic is a conservative extension of classical logic. Classical propositions—as well as predicates as generalized propositions—are viewed as special, *elementary* sorts of games that have no moves and are automatically won by the machine if true, and lost if false. The languages of various reasonably expressive fragments of computability logic would typically include two sorts of atoms: *elementary* atoms p, q, $r(x)$, $s(x,y)$, ...to represent elementary games, and *general atoms* P, Q, $R(x)$, $S(x,y)$, ...to represent any, not-necessarily elementary, games. The classically shaped operators $\neg$, $\wedge$, $\vee$,$\forall$,$\exists$ are conservative generalizations of the corresponding classical operations from elementary games to all games. That is in the sense that, when applied to elementary games, they again produce elementary games, and their meanings happen to coincide with the classical meanings.

2.1. Constant elementary games

These are two 0-ary "operations", for which we use the same symbols $\top$ and $\bot$ as for the two players. $\top$ is an elementary game automatically won by $\top$, and $\bot$ is an elementary game won by $\bot$. Just as classical logic, computability logic sees no difference between two true or two false propositions, so that we have "Snow is white" = "$0 = 0$" = $\top$ and "Snow is black" = "$0 = 1$" = $\bot$.

2.2. Negation

Negation $\neg$ is a role-switch operation: $\neg A$ is obtained from A by turning $\top$'s (legal) moves and wins into $\bot$'s (legal) moves and wins, and vice versa. For example, if *Chess* means the game of chess from the point of view of the white player, then $\neg$*Chess* is the same game from the point of view of the black player. And where $0 = 0$ is an elementary game automatically won by $\top$, $\neg 0 = 0$ is an elementary game automatically won by $\bot$—there are no moves to interchange here, so only the winners are interchanged. From this explanation it must be clear that $\neg$, when applied to elementary games (propositions or predicates), indeed acts like classical negation, as promised.

2.3. Choice operations

The choice operations model decision steps in the course of interaction, with disjunction and existential quantifier meaning $\top$'s choices, and conjunction and universal quantifier meaning choices by $\bot$. For instance, where $f(x)$ is a function, $\bigsqcap x \bigsqcup y\Big(y = f(x)\Big)$ is a game in which the first move/choice is by the environment, consisting in specifying a particular value m for x. Such a move, which intuitively can be seen as asking the machine the question "*what is the value of* $f(m)$*?*" brings the game down to the position $\bigsqcup y\Big(y = f(m)\Big)$. The next step is by the machine, which should specify a value n for y, further bringing the game down to the elementary game $n = f(m)$, won by the machine if true and lost if false. $\top$'s move n can thus be seen as answering/claiming that n is the value of $f(m)$. From this explanation it must be clear that $\bigsqcap x \bigsqcup y\Big(y = f(x)\Big)$ represents the problem of computing f, with $\top$ having an algorithmic winning strategy for this game iff f is a computable function. Similarly, where $p(x)$ is a predicate, $\bigsqcap x\Big(p(x) \sqcup \neg p(x)\Big)$ represents the problem of deciding $p(x)$: here, again, the first move is by the environment, consisting in choosing a value m for x (asking whether $p(m)$ is true); and the next step is by the machine which, in order to win, should choose the true disjunct of $p(m) \sqcup \neg p(m)$, i.e., correctly answer the question. Formally, $A \sqcup B$ can be defined as $\neg(\neg A \sqcap \neg B)$, or $A \sqcap B$ can be defined as $\neg(\neg A \sqcup \neg B)$; furthermore, assuming that the universe of discourse is $\{1,2,3,\ldots\}$, $\bigsqcap x A(x)$ can be defined as $A(1) \sqcap A(2) \sqcap A(3) \sqcap \ldots$ and $\bigsqcup x A(x)$ as $A(1) \sqcup A(2) \sqcup A(3) \sqcup \ldots$ It should be mentioned that making an initial choice of a component by the corresponding player in a choice combination of games is not only that player's privilege, but also an obligation: the player will be considered the loser if it fails to make a choice.

2.4. Parallel operations

The parallel operations combine games in a way that corresponds to the intuition of concurrent computations. Playing $A \wedge B$ or $A \vee B$ means playing, in parallel, the two games A and B. In $A \wedge B$, $\top$ is considered the winner if it wins in both of the components, while in $A \vee B$ it is sufficient to win in one of the components. Then the parallel quantifiers and recurrences are defined by:

$$\begin{aligned}
\bigwedge x A(x) &= A(1) \wedge A(2) \wedge A(3) \wedge \ldots \\
\bigvee x A(x) &= A(1) \vee A(2) \vee A(3) \vee \ldots \\
\curlywedge A &= A \wedge A \wedge A \wedge \ldots \\
\curlyvee A &= A \vee A \vee A \vee \ldots
\end{aligned}$$

To appreciate the difference between choice operations and their parallel counterparts, let us compare the games $Chess \vee \neg Chess$ and $Chess \sqcup \neg Chess$. The former is, in fact, a simultaneous play on two boards, where on the left board $\top$ plays white, and on the right board plays black. There is a simple strategy for $\top$ that guarantees success against any adversary. All that $\top$ needs to do is to mimic, in *Chess*, the moves made by $\bot$ in $\neg Chess$, and vice versa. On the other hand, to win the game $Chess \sqcup \neg Chess$ is not easy: here, at the very beginning, $\top$ has to choose between *Chess* and $\neg Chess$ and then win the chosen one-board game.

While all classical tautologies automatically hold when the classically shaped operators are applied to elementary games, in the general (non-elementary) case the class of valid principles shrinks. For example, $\neg P \vee (P \wedge P)$ is no longer valid. The above "mimicking strategy" would obviously fail in the three-board game

$$\neg Chess \vee (Chess \wedge Chess),$$

for here the best that $\top$ can do is to pair $\neg Chess$ with one of the two conjuncts of $Chess \wedge Chess$. It is possible that then $\neg Chess$ and the unmatched *Chess* are both lost, in which case the whole game will be lost. As much as this example may remind us of linear logic, it should be noted that the class of principles with parallel connectives validated by computability logic is not the same as the class of multiplicative formulas provable in linear or affine logic. An example separating CL from both linear and affine logics is Blass's [3] principle

$$\Big((\neg P \vee \neg Q) \wedge (\neg R \vee \neg S)\Big) \vee \Big((P \vee R) \wedge (Q \vee S)\Big),$$

not provable in affine logic but valid in CL. The same applies to principles containing choice ("additive") and recurrence ("exponential") operators.

2.5. Reduction

The operation $\rightarrow$, defined in the standard way by $A \rightarrow B = \neg A \vee B$, is perhaps most interesting from the computability-theoretic point of view. Intuitively, $A \rightarrow B$ is the problem of *reducing* B to A. Putting it in other words, solving $A \rightarrow B$ means solving B having A as an (external) *computational resource*. "Computational resource" is symmetric to "computational problem": what is a problem (task) for the machine, is a resource for the environment, and vice versa. To get a feel of $\rightarrow$ as a problem reduction operator, let us look at reducing the acceptance problem to the halting problem. The halting problem can be expressed by

$$\sqcap x \sqcap y\Big(Halts(x,y) \sqcup \neg Halts(x,y)\Big),$$

where $Halts(x,y)$ is the predicate "Turing machine (encoded by) x halts on input y". And the acceptance problem can be expressed by

$$\sqcap x \sqcap y\Big(Accepts(x,y) \sqcup \neg Accepts(x,y)\Big),$$

with $Accepts(x,y)$ meaning "Turing machine x accepts input y". While the acceptance problem is not decidable, it is algorithmically reducible to the halting problem. In particular, there is a machine that always wins the game

$$\sqcap x \sqcap y\Big(Halts(x,y) \sqcup \neg Halts(x,y)\Big) \rightarrow \sqcap x \sqcap y\Big(Accepts(x,y) \sqcup \neg Accepts(x,y)\Big).$$

A strategy for solving this problem is to wait till the environment specifies values m and n for x and y in the consequent, thus asking $\top$ the question "does machine m accept input n?". In response, $\top$ selects the same values m and n for x and y in the antecedent (where the roles of $\top$ and $\bot$ are switched), thus asking the counterquestion "does m halt on n?". The environment will have to correctly answer this counterquestion, or else it loses. If it answers "No", then $\top$ also says "No" in the consequent, i.e., selects the right disjunct there, as not halting implies not accepting. Otherwise, if the environment's response in the antecedent is "Yes", $\top$ simulates machine m on input n until it halts and then selects, in the consequent, the left or the right disjunct depending on whether the simulation accepted or rejected.

2.6. Blind operations

The blind group of operations comprises $\forall$ and its dual $\exists$ ($\exists x = \neg \forall x \neg$). The meaning of $\forall x A(x)$ is similar to that of $\sqcap x A(x)$, with the difference that the particular value of x that the environment "selects" is invisible to the machine, so that it has to play blindly in a way that guarantees success no matter what that value is. This way, $\forall$ and $\exists$ produce games with *imperfect information*.

Compare the problems

$$\sqcap x\Big(Even(x) \sqcup Odd(x)\Big)$$

and

$$\forall x\Big(Even(x) \sqcup Odd(x)\Big).$$

Both of them are about telling whether a given number is even or odd; the difference is only in whether that "given number" is communicated to the machine or not. The first problem is an easy-to-win, two-move-deep game of a structure that we have already seen. The second game, on the other hand, is one-move deep with only by the machine to make a move—select the "true" disjunct, which is hardly possible to do as the value of x remains unspecified.

As an example of a solvable non-elementary $\forall$-problem, let us look at

$$\forall x\Big(Even(x) \sqcup Odd(x) \;\rightarrow\; \sqcap y\Big(Even(x+y) \sqcup Odd(x+y)\Big)\Big),$$

solving which means solving what follows "$\forall x$" without knowing the value of x. Unlike $\forall x\Big(Even(x) \sqcup Odd(x)\Big)$, this game is certainly winnable: The machine waits till the environment selects a value n for y in the consequent and also selects one of the $\sqcup$-disjuncts in the antecedent (if either selection is never made, the machine automatically wins). Then: If n is even, in the consequent the machine makes the same selection *left* or *right* as the environment made in the antecedent, and otherwise, if n is odd, it reverses the environment's selection.

2.7. Sequential operations

The new, sequential group of operations forms another natural phylum in this zoo of game operations. The sequential conjunction $A \triangle B$ is a game that starts and proceeds as a play of A; it will also end as an ordinary play of A unless, at some point, $\bot$ decides—by making a special *switch* move—to abandon A and switch to B. In such a case the play restarts, continues and ends as an ordinary play of B without the possibility to go back to A. $A \triangledown B$ is the same, only here it is $\top$ who decides whether and when to switch from A to B. These generalize to the infinite cases $A_0 \triangle A_1 \triangle A_2 \triangle \ldots$ and $A_0 \triangledown A_1 \triangledown A_2 \triangledown \ldots$: here the corresponding player can make any finite number n of switches, in which case the winner in the play will be the player who wins in A_n; and if an infinite number of switches are made, then the player responsible for this is considered the loser. The sequential quantifiers, as we may guess, are defined by

$$\triangle x A(x) = A(1) \triangle A(2) \triangle A(3) \triangle \ldots$$

and

$$\triangledown x A(x) = A(1) \triangledown A(2) \triangledown A(3) \triangledown \ldots,$$

and the sequential recurrence and corecurrence are defined by

$$\barwedge A = A \triangle A \triangle A \triangle \ldots$$

and

$$\barvee A = A \triangledown A \triangledown A \triangledown \ldots.$$

Below are a few examples providing insights into the computational intuitions and motivations associated with the sequential operations.

Let $p(x)$ be any predicate. Remember that the game $\sqcap x\Big(\neg p(x) \sqcup p(x)\Big)$ represents the problem of deciding $p(x)$. Then what is represented by $\sqcap x\Big(\neg p(x) \triangledown p(x)\Big)$? If you guessed that this is the problem of *semideciding* $p(x)$, you have guessed right. It is not hard to see that this game has an effective winning strategy by $\top$ iff $p(x)$ is semidecidable (recursively enumerable). Indeed, if $p(x)$ is semidecidable, a winning strategy is to wait until $\bot$ selects a particular m for x, thus bringing the game down to $\neg p(m) \triangledown p(m)$. After that, $\top$ starts looking for a certificate of $p(m)$'s being true. If and when such a certificate is found (meaning that $p(m)$ is indeed true), $\top$ makes a switch move turning $\neg p(m) \triangledown p(m)$ into the true and hence $\top$-won $p(m)$; and if no certificate exists (meaning that $p(m)$ is false), then $\top$ keeps looking for a non-existent certificate forever and thus never makes any moves, meaning that the game ends as $\neg p(m)$, which, again, is a true and hence $\top$-won elementary game. And vice versa: any effective winning strategy for $\sqcap x\Big(\neg p(x) \triangledown p(x)\Big)$ can obviously be seen as a semidecision procedure for $p(x)$, which accepts an input m iff the strategy ever makes a switch move in the scenario where $\bot$'s initial choice of a value for x is m.

Algorithmic solvability (computability) of games has been shown to be closed under modus ponens, as well as the rules "from A and B conclude $A \wedge B$", "from A conclude $\sqcap x A$", "from A conclude $\barwedge A$". In view of these closures, the validity (= "always computability") of the principles discussed below implies certain known facts from the theory of computation. Needless to say, those examples demonstrate how CL can be used as a systematic tool for defining new interesting properties and relations

between computational problems, and not only reproducing already known theorems but also discovering an infinite variety of new facts.

The following formula, later proven—in a stronger form—to be a theorem of our presumably sound and complete (with respect to validity) first-order system **CL11**, implies—in a sense, "expresses"—the well-known fact that, if both a predicate $p(x)$ and its negation $\neg p(x)$ are recursively enumerable, then $p(x)$ is decidable:

$$\sqcap x\Big(\neg p(x)\mathbin{\triangledown} p(x)\Big) \wedge \sqcap x\Big(p(x)\mathbin{\triangledown}\neg p(x)\Big) \rightarrow \sqcap x\Big(p(x) \sqcup \neg p(x)\Big). \tag{1}$$

Actually, the validity of the above formula means something more than just noted: it means that the problem of deciding $p(x)$ is reducible to the ($\wedge$-conjunction of) the problems of semideciding $p(x)$ and $\neg p(x)$. In fact, a reducibility in an even stronger sense (in a sense that has no name) holds, expressed by the following valid formula:

$$\sqcap x\Big(\Big(\neg p(x)\mathbin{\triangledown} p(x)\Big) \wedge \Big((p(x)\mathbin{\triangledown}\neg p(x)\Big) \rightarrow \Big(p(x) \sqcup \neg p(x)\Big)\Big). \tag{2}$$

Computability logic defines computability of a game $A(x)$ as computability of its $\sqcap$-closure, so the prefix $\sqcap x$ can be safely removed in the above formula and, after writing simply "p" instead of "$p(x)$", the validity of (2) means the same as the validity of the following propositional-level formula, provable in our sound and complete propositional system **CL9**

$$(\neg p\mathbin{\triangledown} p) \wedge (p\mathbin{\triangledown}\neg p) \rightarrow p \sqcup \neg p. \tag{3}$$

Furthermore, the above principle is valid not only for predicates (elementary games) but also for all games that we consider, as evidenced by the provability of the following formula in (the sound) **CL9**

$$(\neg P\mathbin{\triangledown} P) \wedge (P\mathbin{\triangledown}\neg P) \rightarrow P \sqcup \neg P. \tag{4}$$

Similarly, formula (1) remains provable in **CL11** and hence valid with $P(x)$ instead of $p(x)$

$$\sqcap x\Big(\neg P(x)\mathbin{\triangledown} P(x)\Big) \wedge \sqcap x\Big(P(x)\mathbin{\triangledown}\neg P(x)\Big) \rightarrow \sqcap x\Big(P(x) \sqcup \neg P(x)\Big). \tag{5}$$

For our next example, remember the relation of *mapping reducibility* (more often called *many-one reducibility*) of a predicate $q(x)$ to a predicate $p(x)$, defined as existence of an effective function f such that, for any n, $q\Big(n\Big)$ is equivalent to $p\Big(f(n)\Big)$. It is not hard to see that this relation holds if and only if the game

$$\sqcap x \sqcup y\Big(\Big(q(x) \rightarrow p(y)\Big) \wedge \Big(p(y) \rightarrow q(x)\Big)\Big),$$

which we abbreviate as $\sqcap x \sqcup y\Big(q(x) \leftrightarrow p(y)\Big)$, has an algorithmic winning strategy by $\top$. In this sense, $\sqcap x \sqcup y\Big(q(x) \leftrightarrow p(y)\Big)$ expresses the problem of mapping reducing $q(x)$ to $p(x)$. Then the validity (that can be established through **CL11**-provability) of the following formula implies the known fact that, if $q(x)$ is mapping reducible to $p(x)$ and $p(x)$ is recursively enumerable, then so is $q(x)$:[2]

$$\sqcap x \sqcup y\Big(q(x) \leftrightarrow p(y)\Big) \wedge \sqcap x\Big(\neg p(x)\mathbin{\triangledown} p(x)\Big) \rightarrow \sqcap x\Big(\neg q(x)\mathbin{\triangledown} q(x)\Big). \tag{6}$$

As in the earlier examples, the validity of (6), in fact, means something even more: it means that the problem of semideciding $q(x)$ is reducible to the ($\wedge$-conjunction of the) problems of mapping reducing $q(x)$ to $p(x)$ and semideciding $p(x)$.

Certain other reducibilities hold only in a sense weaker than the sense captured by $\rightarrow$. We characterized $A \rightarrow B$ as a game where $\top$ can use A as a computational resource: playing in the role of $\bot$ in A, $\top$ can observe how the adversary is solving A and employ that information in its solving B. It is however important to note that only one "copy" of A is available to $\top$ as a resource in $A \rightarrow B$. In many cases, however, more than one runs of A may be necessary. An example of a reduction of this sort is Turing reduction, where the oracle (resource A) can be queried an unlimited number of times. A way to account for the possibility of repeated usage of A is prefixing it with a recurrence operation. In the following two examples a recurrence that suffices is $\luk$, which (just as the other types of recurrences) induces the weak reduction operation $\succ\!\!-$ defined by $A \succ\!\!- B = \luk A \rightarrow B$.

The following formula is valid (and remains so with $P(x,y)$ instead of $p(x,y)$)

$$\sqcap x \sqcap y\Big(\neg p(x,y) \sqcup p(x,y)\Big) \succ\!\!- \sqcap x\Big(\neg \exists y p(x,y)\mathbin{\triangledown} \exists y p(x,y)\Big), \tag{7}$$

meaning that the problem of semideciding a predicate $\exists y p(x,y)$ is $\succ\!\!-$-reducible to the problem of deciding $p(x,y)$. This, in turn, implies the known fact that if $p(x,y)$ is decidable, then $\exists y p(x,y)$ is recursively enumerable. Unlike the earlier cases where we appealed to provability in (the sound) **CL9** or **CL11** in claiming validity, (7) is not a formula of the languages of those systems because it contains $\succ\!\!-$. So, let us verify its validity directly.

Here is $\top$'s strategy for (7), equally good for (and not depending on) any predicate $p(x,y)$. Wait till $\bot$ specifies a value m for x in the consequent, thus bringing the game down to

[2] By the way, the same principle does not hold with "Turing reducible" instead of "mapping reducible".

$$\sqcap x \sqcap y\Big(\neg p(x,y) \sqcup p(x,y)\Big) \succ\!\!- \Big(\neg\exists y p(m,y) \triangledown \exists y p(m,y)\Big).$$

Then, initialize i to 1 and do the following. Specify x and y in the ith copy of the antecedent as m and i, respectively. $\bot$ will have to respond by choosing one of the $\sqcup$-disjuncts in that copy, which now looks like $\neg p(m,i) \sqcup p(m,i)$, or else it loses. If $\bot$ chooses $\neg p(m,i)$, increment i to $i+1$ and repeat the step. Otherwise, if $\bot$ chooses $p(m,i)$, make a switch move in the consequent and rest your case.

Let us see one more example with $\succ\!\!-$ -reducibility. Let $NEQ(x,y)$ be the predicate "Turing machines (encoded by) x and y are not equivalent", with equivalence meaning that the two machines accept exactly the same inputs. This predicate is neither semidecidable nor co-semidecidable. However, the problem of its semideciding $\succ\!\!-$ -reduces to the halting problem. Specifically, $\top$ has an algorithmic winning strategy for the following game:

$$\sqcap z \sqcap t\Big(\neg Halts(z,t) \sqcup Halts(z,t)\Big) \succ\!\!- \sqcap x \sqcap y\Big(\neg NEQ(x,y) \triangledown NEQ(x,y)\Big). \tag{8}$$

A strategy here is to wait till $\bot$ specifies some values m and n for x and y in the consequent, respectively. Then, initialize i to 1 and do the following. Specify z and t as m and i in one yet-unused copy of the antecedent, and as n and i in another yet-unused copy. That is, ask $\bot$ whether m halts on input i and whether n halts on the same input. $\bot$ will have to provide the correct pair of answers, or else it loses.

(1) If the answers are "No,No", increment i to $i+1$ and repeat the step.
(2) If the answers are "Yes,Yes", then simulate both m and n on input i until they halt. If both machines accept or both reject, increment i to $i+1$ and repeat the step. Otherwise, if one accepts and one rejects, make a switch move in the consequent and celebrate victory.
(3) If the answers are "Yes,No", then simulate m on i until it halts. If m rejects i, increment i to $i+1$ and repeat the step. Otherwise, if m accepts i, make a switch move in the consequent and you win.
(4) If the answers are "No,Yes", then simulate n on i until it halts. If n rejects i, increment i to $i+1$ and repeat the step. Otherwise, if n accepts i, make a switch move in the consequent and you win.

For our last example, remember the concept of the Kolmogorov complexity of a given number m, which can be defined as the size of the smallest Turing machine that returns m on input 1. We denote the Kolmogorov complexity of x by $k(x)$. The latter is known to be bounded, not exceeding x itself.[3] Function $k(x)$ is not computable, meaning that $\top$ has no algorithmic winning strategy in

$$\sqcap x \sqsqcup y\Big(y = k(x)\Big).$$

In contrast, the problem

$$\sqcap x \curlyvee \sqsqcup y\Big(y = k(x)\Big)$$

does have an algorithmic solution. Here is one: wait till $\bot$ specifies a value m for x, thus asking "what is the Kolmogorov complexity of m?" and bringing the game down to $\curlyvee \sqsqcup y\Big(y = k(m)\Big)$. Answer that it is m, i.e., specify y as m, and after that start simulating, in parallel, all machines n with $n < m$ on input 1. Whenever you find a machine n that returns m on input 1 and is smaller than any of the previously found such machines, make a switch move and, in the new copy of $\sqsqcup y\Big(y = k(m)\Big)$, specify y as the size (=logarithm) $|n|$ of n. This obviously guarantees success: sooner or later the real Kolmogorov complexity c of m will be reached and named; and, even though the strategy will never be sure that $k(m)$ is not something yet smaller than c, it will never really find a reason to further reconsider its latest claim that $c = k(m)$.

The following game also has an algorithmic winning strategy, describing which is left as an exercise for the reader:

$$\sqcap x \triangledown y\Big(k(x) = (x - y)\Big).$$

2.8. Branching operations

The branching operations come in the form of branching recurrence $\downarrow$ and its dual branching corecurrence $\curlyvee$, which can be defined by $\curlyvee A = \neg\downarrow\neg A$. The two other—parallel and sequential—sorts of recurrences we have already seen, and it might be a good idea to explain $\downarrow$ by comparing it with them.

What is common to all members of the family of (co)recurrence operations is that, when applied to A, they turn it into a game playing which means repeatedly playing A. In terms of resources, recurrence operations generate multiple "copies" of A, thus making A a reusable/recyclable resource. The difference between the various sorts of recurrences is how "reusage" is exactly understood.

Imagine a computer that has a program successfully playing *Chess*. The resource that such a computer provides is obviously something stronger than just *Chess*, for it permits to play *Chess* as many times as the user wishes, while *Chess*, as such,

[3] Well, strictly speaking, this is so only for sufficiently large numbers x. But since only for finitely many (very small) numbers x do we have $k(x) > x$, we may ignore this minor technicality and assume in our treatment that $k(x)$ never exceeds x.

only assumes one play. The simplest operating system would allow to start a session of *Chess*, then—after finishing or abandoning and destroying it—start a new play again, and so on. The game that such a system plays—i.e., the resource that it supports/provides—is the already known to us sequential recurrence $\mathbin{⅄}$*Chess*, which assumes an unbounded number of plays of *Chess* in a sequential fashion. A more advanced operating system, however, would not require to destroy the old sessions before starting new ones; rather, it would allow to run as many parallel sessions as the user needs. This is what is captured by the parallel recurrence $\curlywedge$*Chess*. As a resource, $\curlywedge$*Chess* is obviously stronger than $\mathbin{⅄}$*Chess* as it gives the user more flexibility. But $\curlywedge$ is still not the strongest form of reusage. A really good operating system would not only allow the user to start new sessions of *Chess* without destroying old ones; it would also make it possible to branch/replicate each particular stage of each particular session, i.e., create any number of "copies" of any already reached position of the multiple parallel plays of *Chess*, thus giving the user the possibility to try different continuations from the same position. What corresponds to this intuition is the branching recurrence $\mathbin{⅋}$*Chess*.

Thus, the user of the resource $\mathbin{⅋}A$ does not have to restart A from the very beginning every time it wants to reuse it; rather, it is (essentially) allowed to backtrack to any of the previous—not necessarily starting—positions and try a new continuation from there, thus depriving the adversary of the possibility to reconsider the moves it has already made in that position. This is in fact the type of reusage every purely software resource allows or would allow in the presence of an advanced operating system and unlimited memory: one can start running process A; then fork it at any stage thus creating two threads that have a common past but possibly diverging futures (with the possibility to treat one of the threads as a "backup copy" and preserve it for backtracking purposes); then further fork any of the branches at any time; and so on. The less flexible type of reusage of A assumed by $\curlywedge A$, on the other hand, is closer to what infinitely many autonomous physical resources would naturally offer, such as an unlimited number of independently acting robots each performing task A, or an unlimited number of computers with limited memories, each one only capable of and responsible for running a single thread of process A. Here the effect of replicating/forking an advanced stage of A cannot be achieved unless, by good luck, there are two identical copies of the stage, meaning that the corresponding two robots or computers have so far acted in precisely the same ways. As for $\mathbin{⅄}A$, it models the task performed by a single reusable physical resource—the resource that can perform task A over and over again any number of times.

A formal definition of branching recurrence is more complicated than the definitions of its parallel and sequential counterparts. For this reason, in our present relaxed tour we refrain from going into more technical details of how, exactly, games of the form $\mathbin{⅋}A$ are played. Such details, together with formal definitions and additional explanations, can be found, for example, in [16]. $\mathbin{⅋}$ also has a series of weaker versions obtained by imposing various restrictions on the quantity and form of reusages. Among the interesting and natural weakenings of $\mathbin{⅋}$ is the *countable branching recurrence* $\mathbin{⅋}^{\aleph_0}$ in the style of Blass's [2,3] *repetition operation R*. See [17] for a discussion of such operations.

Branching recurrence $\mathbin{⅋}$ stands out as the strongest of all recurrence operations, allowing to reuse A (in $\mathbin{⅋}A$) in the strongest algorithmic sense possible. This makes the associated reduction operation $\multimap$, defined by $A \multimap B = \mathbin{⅋}A \to B$, the weakest and hence most general form of algorithmic reduction. The well-known concept of *Turing reduction* has the same claims. The latter, however, is only defined for the traditional, non-interactive sorts of computational problems—two-step, input-output, question-answer sorts of problems that in our terms are written as $\sqcap x\big(p(x) \sqcup \neg p(x)\big)$ (the problem of deciding predicate p) or $\sqcap x \sqcup y\big(y = f(x)\big)$ (the problem of computing function f). And it is no surprise that our $\multimap$, when restricted to such problems, turns out to be equivalent to Turing reduction. Furthermore, when A and B are traditional sorts of problems, $A \multimap B$ further turns out to be equivalent to $A \succ B$ (but not $A \rightarrowtriangle B$), as the differences between $A \succ B$ and $A \multimap B$, while substantial in the general (truly interactive) case, turn out to be too subtle to be relevant when A is a game that models only a very short and simple potential dialogue between the interacting parties, consisting in just asking a question and giving an answer. The benefits from the greater degree of resource-reusage flexibility offered by $A \multimap B$ (as opposed to $A \succ B$) are related to the possibility for the machine to try different reactions to the same action(s) by the environment in A. But such potential benefits cannot be realized when A is, say, $\sqcap x\big(p(x) \sqcup \neg p(x)\big)$. Because here a given individual session of A immediately ends with an environment's move, to which the machine simply has no legal or meaningful responses at all, let alone having multiple possible responses to experiment with.

Thus, both $\multimap$ and $\succ$ are conservative extensions of Turing reduction from traditional sorts of problems to problems of arbitrary degrees and forms of interactivity. Of these two operations, however, only $\multimap$ has the moral right to be called a legitimate successor of Turing reducibility, in the sense that, just like Turing reducibility (in its limited context), $\multimap$ rather than $\succ$ is an ultimate formal counterpart of our most general intuition of algorithmic reduction. And perhaps it is no accident that, as shown in [11,14], its logical behavior—along with the choice operations—is precisely captured by Heyting's intuitionistic calculus. As an aside, this means that CL offers a good justification—in the form of a mathematically strict and intuitively convincing semantics—of the constructivistic claims of intuitionistic logic, and a materialization of Kolmogorov's [18] well known yet so far rather abstract thesis, according to which intuitionistic logic is a logic of problems.

Our recurrence operations, in their logical spirit, are reminiscent of the exponential operators of linear logic. It should be noted that, as shown in [16], linear—in fact, affine—logic turns out to be sound but incomplete when its additives are read as our choice operators, multiplicatives as parallel operators, and exponentials as either parallel or branching recurrences. Here the sequential sort of recurrences stands out in that linear logic becomes simply unsound if its exponentials !,? are interpreted as our $\mathbin{⅄}$,$\curlyvee$.

Just like the acceptance problem $\sqcap x \sqcap y\big(Accepts(x,y) \sqcup \neg Accepts(x,y)\big)$, the Kolmogorov complexity problem $\sqcap x \sqcup y\big(y = k(x)\big)$ is known to be algorithmically reducible—specifically, Turing reducible—to the halting problem. Unlike the former case, however, the reduction in the latter case essentially requires repeated usage of the halting problem as a resource. That is, the reducibility holds only in the sense of $\circ\!\!-$ or $>\!\!-$ but not in the sense of $\rightarrow$. As an exercise, the reader may try to come up with an informal description of an algorithmic winning strategy for either one of the following games:

$$\sqcap x \sqcap y\big(Halts(x,y) \sqcup \neg Halts(x,y)\big) > \!\!- \sqcap x \sqcup y\big(y = k(x)\big);$$
$$\sqcap x \sqcap y\big(Halts(x,y) \sqcup \neg Halts(x,y)\big) \circ \!\!- \sqcap x \sqcup y\big(y = k(x)\big).$$

3. Sequential operators in CL-based applied systems

As we had a chance to see, CL offers a flexible and convenient formalism for specifying and studying computational problems and relations between them. It is a formal theory of computability in the same sense as classical logic is a formal theory of truth, and axiomatizations of various fragments of it provide a systematic way to answer the fundamental question 'what can be computed'. CL also takes us one step closer to developing the long-overdue comprehensive theories of interactive computation and interactive complexity. In fact, if and when further advanced and sufficiently developed (but certainly not in its present, embryonic form), computability logic itself can be considered such a theory or, at least, an integral part of it.

The significance of CL, however, is not limited to theory of computing or pure logic. All of the soundness results for the known axiomatizations of CL come in the strong form that we call *uniform-constructive soundness*. The uniform-constructive soundness of a deductive system means that: (*uniform soundness*:) for every provable formula F, there is a *uniform*, meaning-independent strategy in the sense that it wins the game represented by F no matter how its atoms are interpreted,[4] and (*constructive soundness*:) such a strategy can be effectively extracted from a proof of F. This is good news, signifying that CL is not only about "*what* can be computed", but also equally about "*how* can be computed", opening various application areas, such as (constructive) applied theories, (interactive) knowledgebase systems, or (resource-oriented) AI systems for planning. All such systems would follow the same general scheme. One takes a basic set of formulas expressing problems (computational, informational or physical resources) whose solutions are available (known, maintainable, providable). To such a set S, depending on the context, we may refer as the set of *axioms*, or the *knowledgebase*, or the *resourcebase*. Provability of a formula F in the system can be defined as provability—in pure CL—of the formula $S \rightarrow F$, with S here identified with the $\wedge$-conjunction of its elements.[5] Such a system becomes a problem-solving tool: all one needs for solving a problem is to express it in the language of the system and then find a proof of it. In view of the uniform-constructive soundness of the underlying axiomatization of CL and the closure of computability under modus ponens, a solution for the problem can be automatically obtained from its proof. This is a very brief summary. See, for example, Section 10 of [16] for an extended discussion of CL-based applied systems. In this section, we only outline—very briefly and informally—some intuitions associated with sequential operators that are relevant to potential applications in knowledgebase and planning systems.

In knowledgebase systems, sequential operators can be used to express dynamic or unstable knowledge. Imagine a knowledgebase system that maintains information on all people. Part of the information provided by such a system is knowledge of whether any given person is dead or alive. What the old (sequential-operator-free) language of CL could offer to express such knowledge, with x ranging over people, is

$$⅄ \sqcap x\big(Alive(x) \sqcup Dead(x)\big) : \tag{9}$$

a system containing such a formula/resource in its knowledgebase is able to repeatedly tell us, for any person, whether he or she is alive or dead. This is sufficient to represent a snapshot of some stage of the knowledge(base). Facts change over time though and, in particular, so does the alive/dead status of a person. Yet, once a system asserts $Alive(Tom)$ in the process of playing (9), it cannot take it back later, specifically, when Tom dies. So, (9) is not an adequate way to express a dynamic informational resource of people's alive/dead status. What does fit the bill is

$$⅄ \sqcap x\big(Alive(x) \triangledown Dead(x)\big)$$

instead (but, note, by no means $⅄ \sqcap x(Dead(x) \triangledown Alive(x))$).

Imagine further that the system maintains dynamic information on everybody's marital status. Unlike the alive/dead status, the marital status may change many times through a person's lifetime. To account for having this informational resource, we would include the following formula in the system's knowledgebase

$$⅄ \sqcap x \barwedge\big(Single(x) \triangledown Married(x)\big).$$

[4] As opposed simple soundness which means existence of a winning strategy for each particular interpretation, so that different interpretations may require different strategies.

[5] Of course, this is not the only way to construct CL-based systems. But we are trying to keep things as simple as possible in this brief discussion.

Of course, in both of the above examples, the prefix $\curlywedge\sqcap x$ can be replaced by $\curlyvee\sqcap x$ or just $\bigwedge x$. This would not change the strength of the knowledgebase, but taking $\bigwedge x$ instead of $\curlywedge\sqcap x$ or $\curlyvee\sqcap x$ could negatively affect its efficiency, obligating the system to resolve each and every person's status no matter whether so requested or not.

Having sequential operators in CL-based planning systems could be even more imperative, as such systems are inherently dynamic, where the truth status of various facts keeps changing from situation to situation, and is affected not only by the environment (as in dynamic knowledgebase systems) but also by actions of the agent. Here we restrict ourselves to just one simple and naive example to provide some insights.

Imagine a controller for the outside front entrance light, whose job is to turn the light on at night and—to save energy—turn it off during the daytime. The controller has the capability to repeatedly turn the light on and off. This capability, as a (physical) resource, can be expressed by the formula $\curlywedge(\mathit{Off}\triangle\mathit{On})$, with *Off* expressing the fact "the light is off", and *On* expressing "the light is on". It further has a bright-light sensor, reporting whether it is day or night (or rather whether it is bright enough or not quite so). Let *Day* mean "it is bright enough" and *Night* mean the opposite. The resource provided by such a sensor can then be expressed by $\curlyvee(\mathit{Day}\triangledown\mathit{Night})$. And the goal of the controller is to maintain the truth of $(\mathit{Day}\wedge\mathit{Off})\vee(\mathit{Night}\wedge\mathit{On})$. The overall planning/maintainance problem can then be expressed by

$$\curlyvee(\mathit{Day}\triangledown\mathit{Night})\wedge\curlywedge(\mathit{Off}\triangle\mathit{On})\rightarrow(\mathit{Day}\wedge\mathit{Off})\vee(\mathit{Night}\wedge\mathit{On}). \tag{10}$$

Can the controller, with perfect knowledge of CL and without any other specific knowledge of the world (namely, without knowledge of the meanings of the atoms of (10)), successfully perform its job? With a little thought, this question can be seen to be equivalent to whether there is an effective winning strategy for (10). Sure there is one: every time the environment switches from *Day* to *Night*, switch from *Off* to *On*; and every time the environment switches to the next $\curlyvee$-component in the left conjunct of the antecedent, switch to the next $\curlywedge$-component in the right conjunct. In this example, the *resourcebase* of the planning agent is $\{\curlyvee(\mathit{Day}\triangledown\mathit{Night}),\curlywedge(\mathit{Off}\triangle\mathit{On})\}$, and the *goal* task is $(\mathit{Day}\wedge\mathit{Off})\vee(\mathit{Night}\wedge\mathit{On})$.

Abstract resource semantics, briefly discussed in Appendix A, potentially offers an alternative (similar but not the same) way of using the formalism of computability logic in planning systems.

4. Formal definitions

Here we only provide formal definitions for sequential operations, because they have never been defined before. Strict definitions of the other game operations, as well as definitions of games and all related basic concepts can be found, for example, in [16]. In fact, in what follows we rely on [16] as an external source. Although long, the latter is very easy to read and has a convenient glossary to look up any unfamiliar terms and symbols. A reader not familiar with [16] or unwilling to do some parallel reading, may want to either stop here or just browse the rest of the paper without attempting to go into the technical details of formal definitions and proofs. Due to the very dynamic recent development, computability logic has already reached a point where it is no longer feasible to reintroduce all relevant concepts all over again in each new paper on the subject.

We fix § as a special-meaning symbol, and say that a run Γ is **presequential** iff every move of Γ is either § (we call such moves **switch moves**, or simply **switches**) or $.\alpha$ (we call such moves **non-switch moves**) for some string α. Where $\wp\in\{\top,\bot\}$, by the $\wp$**-degree** of such a run we mean the number of switch moves made by player $\wp$ in it, if this number is finite; if Γ has infinitely many switches by $\wp$, then we say that its $\wp$-degree is infinite. When $\Gamma=\langle\Phi,\wp.\alpha,\Delta\rangle$, by the **degree** of the (indicated occurrence of the) non-switch labmove[6] $\wp.\alpha$ we mean the $\wp$-degree of Φ. Where $i\geq 0$, by $\Gamma^{\#i}$ we mean the result of deleting from Γ all labmoves except the non-switch labmoves of degree i, and then further deleting the prefix "." in each such labmove. For example, we have:

$$\begin{array}{lcl}
\langle\top.\alpha,\bot.\beta,\top\S,\top.\gamma,\bot.\delta,\bot\S,\top.\sigma,\bot.\omega,\top\S,\top.\psi\rangle^{\#0} & = & \langle\top\alpha,\bot\beta,\bot\delta\rangle;\\
\langle\top.\alpha,\bot.\beta,\top\S,\top.\gamma,\bot.\delta,\bot\S,\top.\sigma,\bot.\omega,\top\S,\top.\psi\rangle^{\#1} & = & \langle\top\gamma,\top\sigma,\bot\omega\rangle;\\
\langle\top.\alpha,\bot.\beta,\top\S,\top.\gamma,\bot.\delta,\bot\S,\top.\sigma,\bot.\omega,\top\S,\top.\psi\rangle^{\#2} & = & \langle\top\psi\rangle;\\
\langle\top.\alpha,\bot.\beta,\top\S,\top.\gamma,\bot.\delta,\bot\S,\top.\sigma,\bot.\omega,\top\S,\top.\psi\rangle^{\#3} & = & \langle\rangle.
\end{array}$$

Definition 4.1. Let $A_0,\ldots,A_n$ $(n\geq 1)$ be any constant games. We define the games $A_0\triangle\ldots\triangle A_n$ and $A_0\triangledown\ldots\triangledown A_n$ as follows:

(1)
- A position Φ is a legal position of $A_0\triangle\ldots\triangle A_n$ iff Φ is presequential, the $\bot$-degree of Φ does not exceed n, the $\top$-degree of Φ or any of its initial segments does not exceed the $\bot$-degree of the same position and, for each $i\in\{0,\ldots,n\}$, $\Phi^{\#i}$ is a legal position of A_i.
- Let Γ be a legal run of $A_0\triangle\ldots\triangle A_n$, and k be the $\bot$-degree of Γ. Then Γ is a $\bot$-won run of $A_0\triangle\ldots\triangle A_n$ iff $\Gamma^{\#k}$ is a $\bot$-won run of A_k.

[6] Remember from [16] that *labmove* means "labeled move", i.e., a move prefixed with $\top$ or $\bot$, with such a prefix indicating who has made the move. Terminologically we are not always strict about differentiating between moves and labmoves, and often say "move" where, strictly speaking, we should have said "labmove".

(2) • A position Φ is a legal position of $A_0\triangledown\ldots\triangledown A_n$ iff Φ is presequential, the $\top$-degree of Φ does not exceed n, the $\bot$-degree of Φ or any of its initial segments does not exceed the $\top$-degree of the same position and, for each $i\in\{0,\ldots,n\}$, $\Phi^{\#i}$ is a legal position of A_i.
• Let Γ be a legal run of $A_0\triangledown\ldots\triangledown A_n$, and k be the $\top$-degree of Γ. Then Γ is a $\top$-won run of $A_0\triangle\ldots\triangle A_n$ iff $\Gamma^{\#k}$ is a $\top$-won run of A_k.

Thus, whenever $\bot$ wants to switch from a given component A_i to A_{i+1} in $A_0\triangle\ldots\triangle A_n$, it makes the move §. But, as we see, $\top$, too, is expected to make switch moves in a $\triangle$-game to "catch up" with $\bot$.[7] The switches made by $\bot$ in a $\triangle$-game we call **leading switches**, and the switches made by $\top$ in a $\triangle$-game we call **catch-up switches**. As for a non-switch move $.\alpha$ by either player $\wp$, its effect is making move α in A_k, where k is the number of switch moves made by $\wp$ so far. $A_0\triangledown\ldots\triangledown A_n$, of course, is symmetric. Specifically, here it is $\top$ who makes leading switches, while the switches by $\bot$ are catch-up switches. In either case, the number of leading switches cannot exceed n, and the number of catch-up switches cannot exceed the number of leading switches.

Intuitively, in a play (run) Γ over a sequential combination of games, $\Gamma^{\#i}$ is the sequence of moves made within the ith component (starting the count from 0 rather than 1) of the combination. Each switch move by a player $\wp$ "activates" the next component for that player, in the sense that every subsequent non-switch move $.\alpha$ (until the next switch) by $\wp$ will **signify** making move α in that component; we also say that the **effect** of $.\alpha$ is making move α in the corresponding component. This intuitive and semiformal terminology, on which we will heavily rely in our further treatments, extends to more complex situations and other types of moves as well. Consider, for example, the game $(A\sqcup B)\wedge\Big((C\sqcap D)\triangle(E\sqcap F)\Big)$ and the legal run $\langle\top 1.1,\bot 2.\S,\bot 2..2\rangle$ of it. We say that:

- The effect of the move 1.1 by $\top$ is (or such a move signifies) choosing A within the $A\sqcup B$ component. Indeed, notice that after this move is made in $(A\sqcup B)\wedge((C\sqcap D)\triangle(E\sqcap F))$, the game is brought down to—in the sense that it continues as—$A\wedge((C\sqcap D)\triangle(E\sqcap F))$.
- The effect of the next move 2.§ by $\bot$ is switching from $C\sqcap D$ to $E\sqcap F$ in the $(C\sqcap D)\triangle(E\sqcap F)$ component.
- The effect of the last move 2..2 by $\bot$ is choosing F in the $(E\sqcap F)$ component. If this move was made before the switch, then its effect would be choosing D in the $(C\sqcap D)$ component.

Definition 4.1 extends from finite cases to the infinite case as follows:

Definition 4.2. Let $A_0,A_1,A_2,\ldots$ be any constant games. We define the games $A_0\triangle A_1\triangle A_2\triangle\ldots$ and $A_0\triangledown A_1\triangledown A_2\triangledown\ldots$ as follows:

(1) • A position Φ is a legal position of $A_0\triangle A_1\triangle A_2\triangle\ldots$ iff Φ is presequential, the $\top$-degree of Φ or any of its initial segments does not exceed the $\bot$-degree of the same position and, for each $i\geq 0$, $\Phi^{\#i}$ is a legal position of A_i.
• Let Γ be a legal run of $A_0\triangle A_1\triangle A_2\triangle\ldots$. Then Γ is a $\bot$-won run of $A_0\triangle A_1\triangle A_2\triangle\ldots$ iff the $\bot$-degree of Γ is finite and, where k is that $\bot$-degree, $\Gamma^{\#k}$ is a $\bot$-won run of A_k.
(2) • A position Φ is a legal position of $A_0\triangledown A_1\triangledown A_2\triangledown\ldots$ iff Φ is presequential, the $\bot$-degree of Φ or any of its initial segments does not exceed the $\top$-degree of the same position and, for each $i\geq 0$, $\Phi^{\#i}$ is a legal position of A_i.
• Let Γ be a legal run of $A_0\triangledown A_1\triangledown A_2\triangledown\ldots$. Then Γ is a $\top$-won run of $A_0\triangledown A_1\triangledown A_2\triangledown\ldots$ iff the $\top$-degree of Γ is finite and, where k is that $\top$-degree, $\Gamma^{\#k}$ is a $\top$-won run of A_k.

Even though the above definitions officially define $\triangle$ and $\triangledown$ only for constant games, they extend to all games in the standard way, as explained in the second paragraph of Section 4 of [16]. Specifically, for any not-necessarily constant games $A_0,\ldots,A_n$, $A_0\triangle\ldots\triangle A_n$ is the unique game such that, for any valuation (assignment of constants to variables) e, we have $e[A_0\triangle\ldots\triangle A_n]=e[A_0]\triangle\ldots\triangle e[A_n]$. Similarly for $\triangledown$ and the infinite cases of $\triangle,\triangledown$. (The meaning of the notation $e[\ldots]$, just as the meanings of any other unfamiliar terms or notations, as already noted, can and should be looked up in [16].)

The remaining sequential operations, as we already know from Section 2.7, are defined as follows:

Definition 4.3. For any games A or $A(x)$:

(1) $\mathop{\triangle}A = A\triangle A\triangle A\triangle\ldots$
(2) $\mathop{\triangledown}A = A\triangledown A\triangledown A\triangledown\ldots$
(3) $\mathop{\triangle}xA(x) = A(1)\triangle A(2)\triangle A(3)\triangle\ldots$
(4) $\mathop{\triangledown}xA(x) = A(1)\triangledown A(2)\triangledown A(3)\triangledown\ldots$

It is not hard to see that the DeMorgan dualities hold for the sequential operations, just as they do for all other groups of operations (parallel, choice, branching, blind). Namely, we have:

$$\neg(A_0\triangle\ldots\triangle A_n) = \neg A_0\triangledown\ldots\triangledown\neg A_n,$$
$$\neg(A_0\triangledown\ldots\triangledown A_n) = \neg A_0\triangle\ldots\triangle\neg A_n,$$

[7] This arrangement is necessary to ensure that the sequential operators do not violate the static property of games.

and similarly for infinite sequential conjunctions and disjunctions, including $\bigwedge A$, $\bigvee A$, $\bigtriangleup xA(x)$, $\bigtriangledown xA(x)$.

Whenever new game operations are introduced, one needs to make sure that they preserve the static property of games, for otherwise many things can go wrong:

Theorem 4.4. *The class of static games is closed under our sequential operations.*

Proof. Given in Appendix B. □

5. Logic CL9

In this section, we introduce the propositional system **CL9**. The building blocks of its language are:

- Infinitely many non-logical **elementary atoms**, for which we use the metavariables p,q,r,s.
- Infinitely many non-logical **general atoms**, for which we use the metavariables P,Q,R,S.
- The 0-ary operators $\top$ and $\bot$. They can as well be called **logical atoms**.
- The unary operator $\neg$.
- The operators $\wedge$, $\vee$, $\sqcap$, $\sqcup$, $\bigtriangleup$, $\bigtriangledown$. Their arities are not fixed and can be any $n \geq 2$.

Formulas, to which we refer as **CL9-formulas**, are built from atoms and operators in the standard way, with the requirement (yielding no loss of expressiveness) that $\neg$ can only be applied to non-logical atoms. A **literal** means L or $\neg L$, where L is an atom. Such a literal is said to be elementary, general, non-logical or logical if L is so. When F is not a non-logical atom, $\neg F$ is understood as an abbreviation defined by

$$\begin{array}{rcl}
\neg\top & = & \bot \\
\neg\bot & = & \top \\
\neg\neg E & = & E \\
\neg(E_1 \wedge \ldots \wedge E_n) & = & \neg E_1 \vee \ldots \vee \neg E_n \\
\neg(E_1 \vee \ldots \vee E_n) & = & \neg E_1 \wedge \ldots \wedge \neg E_n \\
\neg(E_1 \sqcap \ldots \sqcap E_n) & = & \neg E_1 \sqcup \ldots \sqcup \neg E_n \\
\neg(E_1 \sqcup \ldots \sqcup E_n) & = & \neg E_1 \sqcap \ldots \sqcap \neg E_n \\
\neg(E_1 \bigtriangleup \ldots \bigtriangleup E_n) & = & \neg E_1 \bigtriangledown \ldots \bigtriangledown \neg E_n \\
\neg(E_1 \bigtriangledown \ldots \bigtriangledown E_n) & = & \neg E_1 \bigtriangleup \ldots \bigtriangleup \neg E_n
\end{array}$$

Also, if we write $E \rightarrow F$, it is to be understood as an abbreviation of $\neg E \vee F$.

The formulas that do not contain elementary non-logical atoms we call **general-base**, and the formulas that do not contain general atoms we call **elementary-base**. This terminology also extends to the corresponding two fragments of **CL9**; in particular, the *general-base fragment* of **CL9** is the set of all general-base theorems of **CL9**, and the *elementary-base fragment* of **CL9** is the set of all elementary-base theorems of **CL9**.

An **interpretation** for the language of **CL9** a function that sends each non-logical elementary atom to an elementary game, and sends each general atom to any, not-necessarily elementary, static game. This mapping extends to all formulas by letting it respect all logical operators as the corresponding game operations. That is, $\top^* = \top$, $(E \bigtriangleup F)^* = E^* \bigtriangleup F^*$, etc. When $F^* = A$, we say that * **interprets** F **as** A.

A formula F is said to be **valid** iff, for every interpretation *, the game F^* is computable. And F is **uniformly valid** iff there is an HPM $\mathcal{H}$, called a **uniform solution** for F, such that $\mathcal{H}$ wins (computes) F^* for every interpretation *.

A **sequential (sub)formula** is one of the form $F_0 \bigtriangleup \ldots \bigtriangleup F_n$ or $F_0 \bigtriangledown \ldots \bigtriangledown F_n$. We say that F_0 is the **head** of such a (sub)formula, and $F_1, \ldots, F_n$ form its **tail**.

The **capitalization** of a formula is the result of replacing in it every sequential subformula by its head.

A formula is said to be **elementary** iff it is a formula of classical propositional logic, i.e., contains no general atoms and no operators other than $\top$,$\bot$,$\neg$, $\wedge$, $\vee$.

An occurrence of a subformula in a formula is **positive** iff it is not in the scope of $\neg$. Otherwise it is **negative**. According to our conventions regarding the usage of $\neg$, only atoms may have negative occurrences.

A **surface occurrence** is an occurrence that is not in the scope of a choice connective and not in the tail of any sequential subformula.

The **elementarization** of a **CL9**-formula F means the result of replacing in the capitalization of F every surface occurrence of the form $G_1 \sqcap \ldots \sqcap G_n$ by $\top$, every surface occurrence of the form $G_1 \sqcup \ldots \sqcup G_n$ by $\bot$, and every positive surface occurrence of each general literal by $\bot$.

Finally, a formula is said to be **stable** iff its elementarization is a classical tautology; otherwise it is **instable**.

Definition 5.1. With $\mathcal{P} \mapsto F$ meaning "from premise(s) $\mathcal{P}$ conclude F", **CL9** is the system given by the following four rules of inference:[8]

[8] There are no axioms, but the rule of Wait can act as such when the set of its premises is empty.

Wait: $\vec{H} \mapsto F$, where F is stable and $\vec{H}$ is the smallest set of formulas satisfying the following two conditions:

(1) whenever F has a surface occurrence of a subformula $G_1 \sqcap \ldots \sqcap G_n$, for each $i \in \{1,\ldots,n\}$, $\vec{H}$ contains the result of replacing that occurrence in F by G_i;
(2) whenever F has a surface occurrence of a subformula $G_0 \triangle G_1 \triangle \ldots \triangle G_n$, $\vec{H}$ contains the result of replacing that occurrence in F by $G_1 \triangle \ldots \triangle G_n$.[9]

Choose: $H \mapsto F$, where H is the result of replacing in F a surface occurrence of a subformula $G_1 \sqcup \ldots \sqcup G_n$ by G_i for some $i \in \{1,\ldots,n\}$.
Switch: $H \mapsto F$, where H is the result of replacing in F a surface occurrence of a subformula $G_0 \triangledown G_1 \triangledown \ldots \triangledown G_n$ by $G_1 \triangledown \ldots \triangledown G_n$.[9]
Match: $H \mapsto F$, where H is the result of replacing in F two—one positive and one negative—surface occurrences of some general atom by a non-logical elementary atom that does not occur in F.

Example 5.2. The following is a **CL9**-proof of $P \sqcup Q \to P \triangledown Q$:

1. $\neg q \vee q$ (from {} by Wait);
2. $\neg Q \vee Q$ (from 1 by Match);
3. $\neg Q \vee (P \triangledown Q)$ (from 2 by Switch);
4. $\neg p \vee (p \triangledown Q)$ (from {} by Wait);
5. $\neg P \vee (P \triangledown Q)$ (from 4 by Match);
6. $(\neg P \sqcap \neg Q) \vee (P \triangledown Q)$ (from {3,5} by Wait).

On the other hand, $P \triangledown Q \to P \sqcup Q$ is not provable. Indeed, $(\neg P \triangle \neg Q) \vee (P \sqcup Q)$ is instable, so Wait cannot be used at the last step of its derivation. It contains no $\triangledown$, so Switch cannot be used, either. And it has only one surface occurrence of an atom, so Match does not apply. This leaves us with Choose. Then the premise is $(\neg P \triangle \neg Q) \vee P$ or $(\neg P \triangle \neg Q) \vee Q$. In either case, Choose no longer applies. If we are dealing with $(\neg P \triangle \neg Q) \vee Q$, evidently the other three rules do not apply either. And if we are dealing with $(\neg P \triangle \neg Q) \vee P$, then only Match applies, which takes us to the premise $(\neg p \triangle \neg Q) \vee p$. This formula could only be the conclusion of Wait, where the set of premises consists of $\neg Q \vee p$. Now we are stuck with this premise, as it cannot be derived by any of the four rules of **CL9**.

With about an equal amount of effort, where **CL9** $\vdash F$ means "F is provable in **CL9**" and **CL9** $\not\vdash F$ means "F is not provable in **CL9**", one can further verify that:

- **CL9** $\vdash P \triangledown Q \to P \vee Q$;
- **CL9** $\not\vdash P \vee Q \to P \triangledown Q$.

Example 5.3. The following is a **CL9**-proof of $(P \wedge Q) \vee (\neg P \triangle \neg R) \vee (\neg Q \triangle \neg S) \vee (R \sqcup S)$:

1. $(p \wedge q) \vee (\neg p \triangle \neg R) \vee \neg s \vee s$ (from {} by Wait);
2. $(p \wedge q) \vee (\neg p \triangle \neg R) \vee \neg S \vee S$ (from 1 by Match);
3. $(p \wedge q) \vee (\neg p \triangle \neg R) \vee \neg S \vee (R \sqcup S)$ (from 2 by Choose);
4. $(p \wedge q) \vee \neg r \vee (\neg q \triangle \neg S) \vee r$ (from {} by Wait);
5. $(p \wedge q) \vee \neg R \vee (\neg q \triangle \neg S) \vee R$ (from 4 by Match);
6. $(p \wedge q) \vee \neg R \vee (\neg q \triangle \neg S) \vee (R \sqcup S)$ (from 5 by Choose);
7. $(p \wedge q) \vee (\neg p \triangle \neg R) \vee (\neg q \triangle \neg S) \vee (R \sqcup S)$ (from {3,6} by Wait);
8. $(p \wedge Q) \vee (\neg p \triangle \neg R) \vee (\neg Q \triangle \neg S) \vee (R \sqcup S)$ (from 7 by Match);
9. $(P \wedge Q) \vee (\neg P \triangle \neg R) \vee (\neg Q \triangle \neg S) \vee (R \sqcup S)$ (from 8 by Match).

Exercise 5.4. Verify that:

1. **CL9** $\vdash P \vee \neg P$
2. **CL9** $\not\vdash P \sqcup \neg P$
3. **CL9** $\not\vdash P \triangledown \neg P$
4. **CL9** $\not\vdash P \to P \wedge P$
5. **CL9** $\vdash P \to P \sqcap P$
6. **CL9** $\not\vdash P \to P \triangle P$
7. **CL9** $\vdash P \wedge Q \to Q \wedge P$
8. **CL9** $\vdash P \sqcap Q \to Q \sqcap P$
9. **CL9** $\not\vdash P \triangle Q \to Q \triangle P$

Exercise 5.5. Check the **CL9**-provability status of the formulas of Exercise 5.4 with p,q instead of P,Q. Where are you getting differences?

[9] In this definition, if $n = 1$, $G_1 \triangle \ldots \triangle G_n$ or $G_1 \triangledown \ldots \triangledown G_n$ is simply understood as G_1.

Exercise 5.6. Construct **CL9**-proofs of formulas (3) and (4) of Section 2.7. Try to extract winning strategies from your proofs.

Below comes our main theorem. It is simply the later-proven Lemmas 7.1 and 9.1 put together:

Theorem 5.7. **CL9** $\vdash F$ *iff* F *is valid (any* **CL9***-formula* F*). Furthermore:*

(a) *There is an effective procedure that takes a* **CL9***-proof of an arbitrary formula* F *and constructs an HPM* $\mathcal{H}$ *such that, for every interpretation* *, $\mathcal{H}$ *computes* F^*.

(b) *If* **CL9** $\nvdash F$, *then* F^* *is not computable for some interpretation* * *that interprets all elementary atoms of* F *as finitary predicates of arithmetical complexity* Δ_2, *and interprets all general atoms of* F *as problems of the form* $(A^1_1 \sqcup \ldots \sqcup A^1_m) \sqcap \ldots \sqcap (A^m_1 \sqcup \ldots \sqcup A^m_m)$, *where each* A^j_i *is a finitary predicate of arithmetical complexity* Δ_2.

The following facts are immediate corollaries of Theorem 5.7, so we state them without proofs:

Fact 5.8. *A* **CL9***-formula is valid iff it is uniformly valid.*

Fact 5.9. **CL9** *is a conservative extension of classical logic. That is, an elementary formula is provable in* **CL9** *iff it is a classical tautology.*

It is also worth noting that **CL9** is decidable, with a brute force decision algorithm obviously running in polynomial space. Whether there are more efficient algorithms is unknown.

6. Preliminaries for the soundness proof

Our proof of Theorem 5.7 starts here and ends in Section 9. It closely follows the soundness and completeness proof given in [6,7] for the less expressive, $\triangle$,$\triangledown$-free logic **CL2**, but the style here is much more relaxed and schematic, often relying on arguments such as "with some thought, we can see that..." in places where [6,7] would provide a full technical elaboration of such "some thought". The present section contains certain necessary preliminaries for the soundness part of the proof.

6.1. Hyperformulas

In the bottom-up (from conclusion to premises) view, the Match rule introduces two occurrences of some new non-logical elementary atom. For technical convenience, we want to differentiate elementary atoms introduced this way from all other elementary atoms, and also somehow keep track of the exact origin of each such elementary atom q—that is, remember what general atom P was replaced by q when Match was applied. For this purpose, we extend the language of **CL9** by adding to it a new sort of non-logical atoms, called **hybrid**. In particular, each hybrid atom is a pair consisting of a general atom P, called its **general component**, and a non-logical elementary atom q, called its **elementary component**. We denote such a pair by P_q. As we are going to see later, the presence of P_q in a (modified **CL9**-) proof will be an indication of the fact that, in the bottom-up view of proofs, q has been introduced by Match and that, when this happened, the general atom that q replaced was P.

For similar reasons, we do not want to fully forget the earlier components of sequential subformulas when Switch or Wait are applied. Hence, we further modify the language of **CL9** by requiring that, in every sequential (sub)formula, one of the components be **underlined**. With such formulas, applying Switch (in the bottom up view) simply moves the underline to the next component of a $\triangledown$-subformula without deleting the "abandoned" component as was the case in **CL9**. Similarly for the effect of (clause 2 of) Wait on $\triangle$-subformulas. The role of an underline is thus to indicate which component of the sequential subformula would be the head of the corresponding subformula in the corresponding **CL9**-proof.

The formulas of this extended and modified language we call **hyperformulas**. We understand each **CL9**-formula F as the hyperformula (and identify F with such) obtained from F by underlining the first component of every sequential subformula. **CL9**-formulas are thus special cases of hyperformulas where the underlined component of a sequential subformula is always the leftmost component, and where no hybrid atoms are present.

By the **general dehybridization** of a hyperformula F we mean the **CL9**-formula that results from F by replacing in the latter every hybrid atom by its general component, and removing all underlines in sequential subformulas. Where * is an interpretation and F is a hyperformula, we define the game

$$F^*$$

as G^*, where G is the general dehybridization of F. Extending the earlier-established lingo to hyperformulas, for a hyperformula F and an interpretation *, whenever $F^* = A$, we say that * **interprets** F as A.

By a **surface occurrence** of a subexpression in a given hyperformula F we mean an occurrence that is not in the scope of a choice operator, such that, if the subexpression occurs within a component of a sequential subformula, that component is underlined or occurs earlier than (is to the left of) the underlined component.

An **active occurrence** is an occurrence such that, whenever it happens to be within a component of a sequential subformula, that component is underlined. If an occurrence is within a component of a sequential subformula which (the component) is to the left of the underlined component of the same subformula, then we say that the occurrence is **abandoned**.

The terms **positive occurrence** and **negative occurrence** have the same meanings for hyperformulas as for **CL9**-formulas.

As in the case of **CL9**-formulas, an **elementary hyperformula** is one not containing choice operators, sequential operators, general atoms and hybrid atoms. Thus, 'elementary hyperformula' and 'elementary **CL9**-formula' (as well as 'formula of classical propositional logic') mean the same.

We define the **capitalization** of a hyperformula F as the result of replacing in it every sequential subformula by its underlined component, after which all underlines are removed.

The **elementarization**

$$\|F\|$$

of a hyperformula F is the result of replacing, in the capitalization of F, every surface occurrence of the form $G_1 \sqcap \ldots \sqcap G_n$ by $\top$, every surface occurrence of the form $G_1 \sqcup \ldots \sqcup G_n$ by $\bot$, every positive surface occurrence of each general literal by $\bot$, and every surface occurrence of each hybrid atom by the elementary component of that atom.

As in the case of **CL9**-formulas, we say that a hyperformula F is **stable** iff its elementarization $\|F\|$ is a classical tautology; otherwise it is **instable**.

A hyperformula F is said to be **balanced** iff, for every hybrid atom P_q occurring in F, the following two conditions are satisfied:

(1) F has exactly two occurrences of P_q, where one occurrence is positive and the other occurrence is negative, and both occurrences are surface occurrences;
(2) the elementary atom q does not occur in F, nor is it the elementary component of any hybrid atom occurring in F other than P_q.

We say that an active occurrence of a hybrid atom (or the corresponding literal) in a balanced hyperformula is **widowed** iff the other occurrence of the same hybrid atom is abandoned.

6.2. Logic **CL9**$^\circ$

In our soundness proof for **CL9** we will employ a "version" of **CL9** called **CL9**$^\circ$. Unlike **CL9** whose language consists only of formulas, the language of **CL9**$^\circ$ allows any balanced hyperformulas, which we also refer to as **CL9**$^\circ$**-formulas**.

Definition 6.1. Logic **CL9**$^\circ$ is given by the following rules for balanced hyperformulas (below simply referred to as "(sub)formulas"):

Wait$^\circ$: $\vec{H} \mapsto F$, where F is stable and $\vec{H}$ is the smallest set of formulas satisfying the following two conditions:

(1) whenever F has an active surface occurrence of a subformula $G_1 \sqcap \ldots \sqcap G_n$, for each $i \in \{1, \ldots, n\}$, $\vec{H}$ contains the result of replacing that occurrence in F by G_i;
(2) whenever F has an active surface occurrence of a subformula

$$G_0 \triangle \ldots \triangle \underline{G_m} \triangle G_{m+1} \triangle \ldots \triangle G_n,$$

$\vec{H}$ contains the result of replacing that occurrence in F by

$$G_0 \triangle \ldots \triangle G_m \triangle \underline{G_{m+1}} \triangle \ldots \triangle G_n.$$

Choose$^\circ$: $H \mapsto F$, where H is the result of replacing in F an active surface occurrence of a subformula $G_1 \sqcup \ldots \sqcup G_n$ by G_i for some $i \in \{1, \ldots, n\}$.

Switch$^\circ$: $H \mapsto F$, where H is the result of replacing in F an active surface occurrence of a subformula

$$G_0 \triangledown \ldots \triangledown \underline{G_m} \triangledown G_{m+1} \triangledown \ldots \triangledown G_n$$

by

$$G_0 \triangledown \ldots \triangledown G_m \triangledown \underline{G_{m+1}} \triangledown \ldots \triangledown G_n.$$

Match$^\circ$: $H \mapsto F$, where H has two—a positive and a negative—active surface occurrences of some hybrid atom P_q, and F is the result of replacing in H both occurrences by P.

Lemma 6.2. *For any* **CL9***-formula* G, *if* **CL9** $\vdash G$, *then* **CL9**$^\circ \vdash G$.

Furthermore, there is an effective procedure that converts any **CL9***-proof of any formula* G *into a* **CL9**$^\circ$*-proof of* G.

Proof. Consider any **CL9**-proof tree T for G, i.e., a tree every node of which is labeled with a **CL9**-formula that follows by one of the rules of **CL9** from the set of (the labels of) its children, with G sitting at the root. For safety and without loss of generality, we assume that, in the bottom-up view of this proof, Match never introduces an elementary atom that had occurrences in some earlier formulas (but such occurrences later disappeared due to cutting off the heads of sequential subformulas).

We modify T as follows. First, we underline the heads of all sequential subformulas of all formulas of T. Next, for each node F of the tree that is derived from its child H by Match—in particular, where H is the result of replacing in F a positive and a negative active surface occurrences of a general atom P by a non-logical elementary atom q—we replace both occurrences of q by the hybrid atom P_q in H as well as in all of its descendants in the tree. In other words, we turn each application of Match into the corresponding application of Match$^\circ$. Next, we similarly turn each application of Switch into an application of Switch$^\circ$. That is, we modify Switch so that this rule, when moving from a conclusion F to the premise (child) H, simply moves the underline from a given component to the next component without otherwise deleting any components in the corresponding $\triangledown$-subformula (and, of course, the undeleted old components should also be added to the corresponding $\triangledown$-subformulas in all descendants of H in the tree). Finally, we do the same for Wait whenever it modifies a $\triangle$-subformula. It is not hard to see that the resulting tree T° is a **CL9**$^\circ$-proof of G. □

6.3. Perfect interpretations

An interpretation * is said to be **perfect** iff it interprets every atom as a constant game. All of our game operations can be easily seen to preserve the constant property of games (for the non-sequential operations, this was officially established in [5], Theorem 14.1), which means that perfect interpretations interpret all (hyper)formulas as constant games. This fact may be worth marking as we will often implicitly rely on it. For an interpretation * and valuation e, the **perfect interpretation induced by** $(^*,e)$ is the interpretation † that interprets each (elementary or general) atom L as the constant game $e[L^*]$.

Lemma 6.3. *Assume F is any* **CL9**$^\circ$*-formula, e any valuation, * any interpretation and † the perfect interpretation induced by $(^*,e)$. Then $e[F^*] = F^\dagger$.*

Proof. Induction on the complexity of F. For an atomic F, $e[F^*] = F^\dagger$ is immediate. And the inductive step is also straightforward, taking into account that the operations $e[\ldots]$, *, † commute with $\neg, \wedge, \vee, \sqcap, \sqcup, \triangle, \triangledown$. □

6.4. Manageability

We continue using our informal jargon introduced in Section 4. Let us further agree, context permitting, to see no distinction between (hyper)formulas and the games that they would represent once some interpretation was applied to them. We can say, for example, that "$\top$ makes a choice move within the subformula $E \sqcup F$ of $(E \sqcup F) \wedge G$" to mean that $\top$ makes the move 1.1 or 1.2 (thus choosing E or F in the first $\wedge$-conjunct of the formula/game). For terminological simplicity, we may also not always be careful about distinguishing between a subformula and a particular occurrence of it and, for example, say "active (surface, etc.) subformula E of F" instead of "active occurrence of E in F". When we use such terminology, we usually have some run Γ in mind—the run about the moves of which we are talking. Let us call such a run the **contextual run**.

Definition 6.4. Let F be a **CL9**$^\circ$-formula. We say that a run Γ is **F-manageable** iff, with Γ being the contextual run, the following five conditions are satisfied:

(1) No (choice) moves have been made within active choice subformulas.
(2) If $E_0 \triangle \ldots \triangle \underline{E_i} \triangle \ldots \triangle E_n$ is an active subformula of F, both of the players have made exactly i switches in this subformula.[10]
(3) If $E_0 \triangledown \ldots \triangledown \underline{E_i} \triangledown \ldots \triangledown E_n$ is an active subformula of F, $\top$ has made exactly i switches in this subformula, and $\bot$ has made $\leq i$ switches.
(4) No moves have been made by $\top$ within general atoms.
(5) If, for some hybrid atom P_q both of whose occurrences are active, Σ^+ and Σ^- are the sequences of the moves that have been made within the positive and the negative occurrence of P_q in F, respectively, then Σ^+ is a $\top$-delay (see [16], Section 5) of $\neg\Sigma^-$.

The above technical concept will play a central role in our soundness proof for **CL9**. The main intuition here is that, when Γ is F-manageable, playing it in no way affects/modifies the choice subgames of the game (clause 1), guarantees that at any

[10] According to our terminological conventions, making moves in (a given occurrence of) subformula $E_0 \triangle \ldots \triangle E_n$ means making moves in the (sub)game represented by this (sub)formula. Note that such a (sub)game does not depend on which of the components of the formula is underlined. So, in the present context the phrase "this subformula" should be understood as referring to $E_0 \triangle \ldots \triangle E_n$ without regard to where the underline goes. The same applies to clause 3 of the definition. And a similar comment should be made for clause 5, where "within …P_q" can or should be understood as "within …P", as the game represented by P_q does not depend on q or the presence/absence of it.

time the underline in a sequential subformula accurately indicates the number of leading switches made therein (clauses 2 and 3), lets the subgames in the "matched" occurrences of atoms evolve to—in a sense—the same games (clause 5), and makes sure that $\top$ does not make any hasty moves in unmatched atoms (clause 4), so that, if and when at some later point such an atom finds a match, $\top$ will still have a chance to "even out" the corresponding two subgames.

Lemma 6.5. *Let E be any* **CL9**$^\circ$*-formula, * any perfect interpretation, and Γ an infinite run with arbitrarily long finite initial segments that are E-manageable legal positions of E^*. Then Γ is an E-manageable legal run of E^*.*

Proof. Assume the conditions of the lemma. They imply that every finite initial segment of Γ is a legal position of E^*, which (by the definition of "legal run") means that Γ is a legal run of E^*. Also, obviously Γ satisfies conditions 1, 2, 3 and 4 of Definition 6.4 because it has arbitrarily long initial segments that satisfy those conditions. So, what remains to show is that Γ also satisfies condition 5 of Definition 6.4.

Suppose, for a contradiction, that this is not the case. In particular, let Σ^+, Σ^- and P_q be as described in the antecedent of condition 5, and assume that Σ^+ is not a $\top$-delay of $\neg\Sigma^-$. This means that at least one of the following two statements is true:

(i) For one of the players $\wp$, the subsequence of the $\wp$-labeled moves of Σ^+ (i.e., the result of deleting in Σ^+ all $\neg\wp$-labeled moves) is not the same as that of $\neg\Sigma^-$, or
(ii) For some k,n, in $\neg\Sigma^-$ the nth $\top$-labeled move is made later than the kth $\bot$-labeled move, but in Σ^+ the nth $\top$-labeled move is made earlier than the kth $\bot$-labeled move.

Whether (i) or (ii) is the case, it is not hard to see that, beginning from some (finite) m, every initial segment Ψ of Γ of length $\geq m$ will satisfy the same (i) or (ii) in the role of Γ, and hence Ψ will not be an E-manageable position of E^*. This contradicts the assumptions of our lemma. □

Lemma 6.6. *Assume A is a constant static game, $\wp$ is either player, and Γ,Δ are runs such that Δ is a $\wp$-delay of Γ. Then*:

(1) *If Δ is a $\wp$-illegal run of A, then so is Γ.*
(2) *If Γ is a $\neg\wp$-illegal run of A, then so is Δ.*

Proof. The above is a fact known from [5] (Lemma 4.7). □

Below and later, by saying "$\wp$ makes move α in position Ω" we mean that such a move is appended to Ω as a new move. This creates the new position $\langle\Omega,\wp\alpha\rangle$, which serves as the contextual run when talking about the effects of α or other, earlier moves of the position.

Lemma 6.7. *In each of the following clauses, we assume that E is a* **CL9**$^\circ$*-formula, * is a perfect interpretation, and Ω is an E-manageable legal position of E^*.*

1. Suppose $\top$ makes a move α in position Ω, whose effect is choosing the ith disjunct in an active surface occurrence of a subformula $G_1 \sqcup \ldots \sqcup G_n$ in E. Of course, such a move is legal.[11] *Let H be the result of replacing in E the above occurrence by G_i. Then $\langle\Omega\rangle$ is an H-manageable legal position of H^*, and $\langle\Omega,\top\alpha\rangle E^* = \langle\Omega\rangle H^*$.*

2. Suppose $\top$ makes a move α in position Ω, whose effect is making a (leading) switch in an active surface occurrence[12] *of a subformula*

$$G_0 \triangledown \ldots \triangledown \underline{G_i} \triangledown G_{i+1} \triangledown \ldots \triangledown G_n.$$

Of course, such a move is legal. Let H be the result of replacing in E the above occurrence by

$$G_0 \triangledown \ldots \triangledown G_i \triangledown \underline{G_{i+1}} \triangledown \ldots \triangledown G_n.$$

Then $\langle\Omega,\top\alpha\rangle$ is an H-manageable legal position of H^*, and $\langle\Omega,\top\alpha\rangle E^* = \langle\Omega,\top\alpha\rangle H^*$.

3. Suppose H is a **CL9**$^\circ$*-formula, and it results from E by replacing in it a positive active surface occurrence O^+ and a negative active surface occurrence O^- of a general atom P by a hybrid atom P_q. Further assume that $\pi_1,\ldots,\pi_n$ and $\nu_1,\ldots,\nu_m$ are the sequences of moves made so far (during playing Ω, that is) by $\bot$ within O^+ and O^-, respectively. Let Ω' be the result of adding to Ω m moves by $\top$ whose effects are making the moves $\nu_1,\ldots,\nu_m$ in O^+, and further adding to the resulting position n moves by $\top$ whose effects are making the moves $\pi_1,\ldots,\pi_n$ in O^-. Then Ω' is an H-manageable legal position of $H^* = E^*$.*

4. Suppose $\bot$ makes a legal move α in position Ω, whose effect is moving within some abandoned occurrence of a subformula or a widowed occurrence of a hybrid literal. Then $\langle\Omega,\bot\alpha\rangle$ remains an E-manageable legal position of E^*.

[11] Here and in the subsequent clauses, "legal" should be understand with respect to E^*. That is, "α is a legal move made by $\wp$ (in position Ω)" means nothing but that $\langle\Omega,\wp\alpha\rangle$ is a legal position of E^*.
[12] Here and in the subsequent clauses, E is the contextual formula. Specifically, by simply saying "occurrence", we mean occurrence in E.

5. Suppose $\bot$ makes a legal move α in position Ω, whose effect is moving in some active surface occurrence of a general atom. Then $\langle\Omega,\bot\alpha\rangle$ remains an E-manageable legal position of E^*.

6. Suppose $\bot$ makes a legal move α in position Ω, whose effect is making a (catch-up) switch in an active surface occurrence of a subformula

$$G_0\sqcup\ldots\sqcup\underline{G_i}\sqcup G_{i+1}\sqcup\ldots\sqcup G_n.$$

Then $\langle\Omega,\bot\alpha\rangle$ remains an E-manageable legal position of E^*.

7. Suppose $\bot$ makes a legal move α in position Ω, whose effect is making a move γ within an active surface non-widowed occurrence of a hybrid atom. Let β be the move by $\top$ whose effect is making the same move γ within the other active surface occurrence of the same hybrid atom. Then $\langle\Omega,\bot\alpha,\top\beta\rangle$ remains an E-manageable legal position of E^*.

8. Suppose $\bot$ makes a legal move α in position Ω, whose effect is choosing the ith conjunct in an active surface occurrence of a subformula $G_1\sqcap\ldots\sqcap G_n$. Let H be the result of replacing in E the above occurrence by G_i. Then $\langle\Omega\rangle$ is an H-manageable legal position of H^*, and $\langle\Omega,\bot\alpha\rangle E^*=\langle\Omega\rangle H^*$.

9. Suppose $\bot$ makes a legal move α in position Ω, whose effect is making a (leading) switch in an active surface occurrence of a subformula

$$G_0\sqcap\ldots\sqcap\underline{G_i}\sqcap G_{i+1}\sqcap\ldots\sqcap G_n.$$

Let H be the result of replacing in E the above occurrence by

$$G_0\sqcap\ldots\sqcap G_i\sqcap\underline{G_{i+1}}\sqcap\ldots\sqcap G_n.$$

Then $\langle\Omega,\bot\alpha,\top\alpha\rangle$ is an H-manageable legal position of H^*, and $\langle\Omega,\bot\alpha,\top\alpha\rangle E^*=\langle\Omega,\bot\alpha,\top\alpha\rangle H^*$.

10. If $\bot$ makes a legal move α in position Ω, then it satisfies the conditions of one of the clauses 4–9.

Proof. Assume the conditions of the lemma and, in addition, when discussing each clause of the lemma, assume the additional conditions of that clause.

Clause 1. It is not hard to see that each of the five conditions of Definition 6.4 is inherited by H and Ω from E and Ω. This means that $\langle\Omega\rangle$ is H-manageable. The fact $\langle\Omega,\top\alpha\rangle E^*=\langle\Omega\rangle H^*$ is also obvious, as the effect of (the clearly legal) move $\top\alpha$ in $\langle\Omega\rangle E^*$ is turning the $(G_1\sqcup\ldots\sqcup G_n)^*$ component of the latter into G_i^*, i.e., turning E^* into H^*.

Clause 2. As in the previous clause, with a little thought one can see that each of the five conditions of Definition 6.4 is inherited by H and $\langle\Omega,\top\alpha\rangle$ from E and Ω, so $\langle\Omega,\top\alpha\rangle$ is H-manageable. As for $\langle\Omega,\top\alpha\rangle$'s being a legal position of H^* and the fact $\langle\Omega,\top\alpha\rangle E^*=\langle\Omega,\top\alpha\rangle H^*$, it is sufficient to note that we simply have $E^*=H^*$.

Clause 3. Let

$$\Phi^+=\langle\bot\pi_1,\ldots,\bot\pi_n,\top\nu_1,\ldots,\top\nu_m\rangle$$

and

$$\Phi^-=\langle\bot\nu_1,\ldots,\bot\nu_m,\top\pi_1,\ldots,\top\pi_n\rangle.$$

Thus, Φ^+ and Φ^- are the sequences of all moves made by the two players within O^+ and O^-, respectively, while playing Ω' (here the contextual run). Note that

$$\Phi^+ \text{ is a } \top\text{-delay of } \neg\Phi^-. \tag{11}$$

This implies that Ω' is H-manageable, because conditions 1–4 of Definition 6.4 are obviously inherited by H and Ω' from E and Ω, and so is condition 5 for any active hybrid atom R_s different from P_q.

We, of course, have $H^*=E^*$. So, what remains to show is that Ω' is a legal position of E^*. The assumption that Ω is a legal position of E^* obviously implies that:

$$\langle\bot\pi_1,\ldots,\bot\pi_n\rangle \text{ is a legal position of } P^*; \tag{12}$$

$$\langle\bot\nu_1,\ldots,\bot\nu_m\rangle \text{ is a legal position of } \neg P^*. \tag{13}$$

Assume, for a contradiction, that Ω' is not a legal position of E^*. Evidently this can be the case only if either Φ^+ is not a legal position of P^* or Φ^- is not a legal position of $\neg P^*$ (or both).

Suppose Φ^+ is not a legal position of P^*. In view of (12), Φ^+ cannot be a $\bot$-illegal position of P^*. So, it must be $\top$-illegal. But then, by (11) and clause 1 of Lemma 6.6, $\neg\Phi^-$ is a $\top$-illegal position of P^*, meaning that Φ^- is a $\bot$-illegal position of $\neg P^*$. This, however, is in obvious contradiction with (13).

Suppose now Φ^- is not a legal position of $\neg P^*$. This case is similar/symmetric to the previous one. In view of (13), Φ^- cannot be a $\bot$-illegal position of $\neg P^*$. So, it must be $\top$-illegal, which means that $\neg\Phi^-$ is a $\bot$-illegal position of P^*. But then, by (11) and clause 2 of Lemma 6.6, Φ^+ is a $\bot$-illegal position of P^*. This, however, is in contradiction with (12).

Clauses 4,5,6,7 are obvious.

Clause 8 is symmetric to Clause 1.

Clause 9 is similar to Clause 2.

Clause 10 is obvious. □

6.5. Finalization

Note that when a formula E is elementary and an interpretation * is perfect, E^* is one of the two constant elementary games $\top$ or $\bot$. Here we define the relation $\leq$ on constant elementary games by stipulating that $A \leq B$ iff $A = \bot$ or $B = \top$. In other words, $A \leq B$ iff $(A \to B) = \top$.

Lemma 6.8. *Suppose * is a perfect interpretation, F_1 is an elementary formula, and F_2 is the result of replacing in F_1 a positive occurrence of an (elementary) literal L_1 by an elementary (logical or non-logical) literal L_2, such that $L_1^* \leq L_2^*$. Then $F_1^* \leq F_2^*$.*

Proof. The above lemma does nothing but rephrases, in our terms, a known fact from classical logic, according to which, if in an interpreted formula F_1 we replace a positive occurrence of a subformula L_1 by a formula L_2 whose Boolean value is not less than that of L_1 (L_2 is "at least as true" as L_1), then the value of the resulting formula F_2 will not be less than that of F_1 (F_2 will not be "less true" than F_1). □

Now we introduce the operation $\langle\Gamma\rangle\!\downarrow\! A$ of the type

$$\{runs\}\times\{constant\ games\} \to \{constant\ elementary\ games\},$$

which is rather similar to prefixation. Intuitively $\langle\Gamma\rangle\!\downarrow\! A$, that we call the Γ**-finalization** of A, is the proposition "Γ is a $\top$-won run of A". This operation is only defined when Γ is a legal run of A. We agree that, every time we make a statement that applies $\langle\Gamma\rangle\!\downarrow$ to a constant game A, we imply that Γ is a legal run of A and hence $\langle\Gamma\rangle\!\downarrow\! A$ is defined. Here is the formal definition of the operation of finalization:

Definition 6.9. Assume A is a constant game and Γ a legal run of A. Then $\langle\Gamma\rangle\!\downarrow\! A$ is the constant elementary game defined by $\mathbf{Wn}^{\langle\Gamma\rangle\!\downarrow\! A}\langle\rangle = \mathbf{Wn}^A\langle\Gamma\rangle$.

Lemma 6.10. *Assume E is a stable* **CL9**$^\circ$*-formula, * is a perfect interpretation, and Γ is an E-manageable legal run of E^*. Then Γ is a $\top$-won run of E^*.*

Proof. Assume the conditions of the lemma. Throughout this proof, Γ is the contextual run.

For each (positive) active surface occurrence of each general or hybrid literal in E, let us fix an elementary non-logical atom that we call the **surrogate** for that occurrence. We assume that all surrogates are pairwise distinct, and none of them occurs in E (either directly or as the elementary component of a hybrid atom). Since these atoms do not occur in E, we may make an arbitrary assumption regarding how they are interpreted by * (otherwise replace * by an appropriate interpretation). In particular, we assume that, whenever r is the surrogate for an active occurrence O of a general or hybrid literal L, we have $r^* = \langle\Sigma\rangle\!\downarrow\! L^*$, where Σ is the sequence of (lab)moves made within that occurrence.

Let E_1 denote the result of replacing in the capitalization of E:

- every positive surface occurrence of a general or hybrid literal by its surrogate;
- every surface occurrence of a subformula of the form $H_1 \sqcap \ldots \sqcap H_n$ by $\top$;
- every surface occurrence of a subformula of the form $H_1 \sqcup \ldots \sqcup H_n$ by $\bot$.

With some thought, we can see that

$$\langle\Gamma\rangle\!\downarrow\! E^* = E_1^*. \tag{14}$$

Assume E has k active, positive, non-widowed hybrid atoms, where $q^1,\ldots,q^k$ are the elementary components of those atoms and $P^1,\ldots,P^k$ are the corresponding general components. Let $r_1,\ldots,r_k$ be the surrogates for the positive occurrences of $P^1_{q^1},\ldots,P^k_{q^k}$, respectively, and let $\Sigma_1^+,\ldots,\Sigma_k^+$ be the sequences of moves that have been made (while playing Γ) within these k occurrences. Next, let $s_1,\ldots,s_k$ be the surrogates for the occurrences of $\neg P^1_{q^1},\ldots,\neg P^k_{q^k}$, respectively, and let $\Sigma_1^-,\ldots,\Sigma_k^-$ be the sequences of moves that have been made within these k occurrences. Let E_2 be the result of replacing in E_1 each atom s_i ($1 \leq i \leq k$) by $\neg r_i$. According to clause 5 of Definition 6.4, for each $1 \leq i \leq k$, Σ_i^+ is a $\top$-delay of $\neg\Sigma_i^-$. Hence, as P_i^* is a static game, we have $\langle\neg\Sigma_i^-\rangle\!\downarrow\! P_i^* \leq \langle\Sigma_i^+\rangle\!\downarrow\! P_i^*$, which is the same as to say that $\neg(\langle\Sigma_i^+\rangle\!\downarrow\! P_i^*) \leq \neg(\langle\neg\Sigma_i^-\rangle\!\downarrow\! P_i^*)$. But, by the definition of $\neg$, $\neg(\langle\neg\Sigma_i^-\rangle\!\downarrow\! P_i^*) = \langle\Sigma_i^-\rangle\!\downarrow\!\neg P_i^*$. So, $\neg(\langle\Sigma_i^+\rangle\!\downarrow\! P_i^*) \leq \langle\Sigma_i^-\rangle\!\downarrow\!\neg P_i^*$. Now remember that $\langle\Sigma_i^+\rangle\!\downarrow\! P_i^* = r_i^*$ and $\langle\Sigma_i^-\rangle\!\downarrow\!\neg P_i^* = s_i^*$. Thus, $\neg r_i^* \leq s_i^*$. Then, applying Lemma 6.8 k times, we get

$$E_2^* \leq E_1^*. \tag{15}$$

Next, let E_3 be the result of replacing in E_2 each surrogate for a general literal by $\bot$. Applying Lemma 6.8 as many times as the number of such surrogates, we get

$$E_3^* \leq E_2^*. \tag{16}$$

Now compare E_3 with $\|E\|$. An analysis of how these two formulas have been obtained from E can reveal that E_3 is just the result of replacing in $\|E\|$ all (both) occurrences of each atom q_i from the earlier-mentioned list $q_1, \ldots, q_k$ by r_i. That is, E_3 is a substitutional instance of $\|E\|$. The latter is classically valid because, by our assumptions, E is stable. Therefore E_3 is also classically valid, and hence $E_3^* = \top$. Then statements (16), (15) and (14) yield $\langle\Gamma\rangle\downarrow E^* = \top$. In other words, Γ is a $\top$-won run of E^*. □

7. The soundness of CL9

Lemma 7.1. *If* **CL9** $\vdash F$, *then F is valid (any formula F).*

Moreover, there is an effective procedure that takes a **CL9***-proof of an arbitrary formula F and returns an HPM $\mathcal{H}$ such that, for every interpretation *, $\mathcal{H}$ computes F^*.*

Proof. According to Theorem 4.4, our sequential operations preserve the static property of games. The same is known to be true for all other operations studied in computability logic (Theorem 24 of [16]). So, for any formula F and interpretation *, the game F^* is static. Further, it is known that, for static games, HPMs and EPMs have the same computing power, and that, moreover, every EPM can be effectively converted into an HPM such that the latter wins every static game won by the former (Theorem 28 of [16]). Therefore, it would be sufficient to prove the above lemma—in particular, the 'Moreover' clause of it—with "EPM $\mathcal{S}$" instead of "HPM $\mathcal{H}$".

Furthermore, it would be sufficient to restrict interpretations to perfect ones. Indeed, suppose a machine $\mathcal{M}$ (whether it be an EPM or an HPM) wins $F^\dagger$ for every perfect interpretation †, and let * be a not-necessarily perfect interpretation. We want to see that the same machine $\mathcal{M}$ also wins F^*. Suppose this is not the case, i.e., $\mathcal{M}$ loses F^* on some valuation e. This means that, where Γ is the run spelled by some e-computation branch of $\mathcal{M}$, we have $\mathbf{Wn}^{e[F^*]}\langle\Gamma\rangle = \bot$. Now, let † be the perfect interpretation induced by $(^*, e)$. According to Lemma 6.3, $e[F^*] = F^\dagger$. Thus, $\mathbf{Wn}^{F^\dagger}\langle\Gamma\rangle = \bot$, so that $\mathcal{M}$ does not win $F^\dagger$, which is a contradiction.

Finally, Lemma 6.2 allows us to safely replace "**CL9**" by "**CL9**$^\circ$" in our present lemma.

In view of the above observations, Lemma 7.1 is an immediate consequence of the following Lemma 7.2. □

Lemma 7.2. *There is an effective procedure that takes a* **CL9**$^\circ$*-proof of an arbitrary formula F and returns an EPM $\mathcal{S}$ such that, for every perfect interpretation *, $\mathcal{S}$ computes F^*.*

Proof. Every **CL9**$^\circ$-proof, in fact, encodes a valuation- and interpretation-independent winning strategy for $\top$, and the EPM $\mathcal{S}$ that we are going to describe just follows such a strategy.

We provide only a semiformal description of how our strategy/machine $\mathcal{S}$ works for a **CL9**$^\circ$-provable formula F. Restoring suppressed technical details, if necessary, does not present a problem.[13] Fix an arbitrary valuation e and a perfect interpretation *. Since * is perfect, the e parameter is irrelevant and it can always be safely omitted. Furthermore, in concordance with our earlier agreements, we will often omit * as well and write E instead of E^*. That is, we continue abusing terminology and notation by (often) identifying game E^* with formula E.

The strategy that our $\mathcal{S}$ follows is a recursive one, at every step dealing with $\langle\Omega\rangle E^*$, where E is a **CL9**$^\circ$-provable formula and Ω is an E-manageable legal position of E^*. Initially $E = F$ and $\Omega = \langle\rangle$. How $\mathcal{S}$ acts on $\langle\Omega\rangle E^*$ depends on by which of the four rules E is derived in **CL9**$^\circ$ (some **CL9**$^\circ$-proof is assumed to be fixed).

If E is derived by Choose$^\circ$ from H as described in Definition 6.1, $\mathcal{S}$ makes the move α whose effect is choosing G_i in the $G_1 \sqcup \ldots \sqcup G_n$ subformula of E. For example, if $E = (G_1 \sqcup G_2) \wedge (G_3 \sqcap G_4)$ and $H = G_2 \wedge (G_3 \sqcap G_4)$, then '1.2' is such a move α. Clause 1 of Lemma 6.7 tells us that Ω remains an H-manageable legal position of H^* and that $\top\alpha$ brings $\langle\Omega\rangle E^*$ down to $\langle\Omega\rangle H^*$. So, after making move α, $\mathcal{S}$ switches to its strategy for $\langle\Omega\rangle H^*$. This, by the induction hypothesis, guarantees success.

If E is derived by Switch$^\circ$ from H as described in Definition 6.1, then $\mathcal{S}$ makes the move α whose effect is making a switch in the $G_0 \triangledown \ldots \triangledown \underline{G_m} \triangledown G_{m+1} \triangledown \ldots \triangledown G_n$ subformula. For example, if $E = (G_0 \triangledown \underline{G_1} \triangledown G_2) \wedge (\underline{G_3} \triangledown G_4)$ and $H = (G_0 \triangledown G_1 \triangledown \underline{G_2}) \wedge (\underline{G_3} \triangledown G_4)$, then '1.§' is such a move. Clause 2 of Lemma 6.7 tells us that $\langle\Omega, \top\alpha\rangle$ remains an H-manageable legal position of H^* and that $\top\alpha$ brings $\langle\Omega\rangle E^*$ down to $\langle\Omega, \top\alpha\rangle H^*$. So, after making move α, the machine switches to its strategy for $\langle\Omega, \top\alpha\rangle H^*$ and, by the induction hypothesis, wins.

If E is derived by Match$^\circ$ from H through replacing the two (active surface) occurrences of a hybrid atom P_q in H by P, then the machine finds within Ω and copies, in the positive occurrence of P_q, all of the moves made so far by the environment in the negative occurrence of P_q (or rather in the corresponding occurrence of P), and vice versa. This series of moves brings

[13] A similar proof for **CL2** given in [6] was not lazy to go into all technical details, for which reason it was much longer than the present proof.

the game down to $\langle\Omega'\rangle E^* = \langle\Omega'\rangle H^*$, where Ω' is result of adding those moves to Ω. Clause 3 of Lemma 6.7 guarantees that Ω' is an H-manageable legal position of H^*. So, now $\mathcal{S}$ switches to its successful strategy for $\langle\Omega'\rangle H^*$ and eventually wins.

Finally, suppose E is derived by Wait$^\circ$. Our machine keeps granting permission ("waiting"). In view of Lemma 6.10, if $\bot$ never makes a move, $\mathcal{S}$ wins the game. Suppose now $\bot$ makes a move α. This should be a legal move, or else $\mathcal{S}$ automatically wins. Then, according to clause 10 of Lemma 6.7, α should satisfy the conditions of one of the following six cases.

Case 1. α is a move whose effect is moving in some abandoned subformula or a widowed hybrid literal of E. According to clause 4 of Lemma 6.7, $\langle\Omega,\bot\alpha\rangle$ remains an E-manageable legal position of E^*. In this case, $\mathcal{S}$ calls itself on $\langle\Omega,\bot\alpha\rangle E^*$.

Case 2. α is a move whose effect is moving in some active surface occurrence of a general atom in E. According to clause 5 of Lemma 6.7, $\langle\Omega,\bot\alpha\rangle$ remains an E-manageable legal position of E^*. Again, in this case, $\mathcal{S}$ calls itself on $\langle\Omega,\bot\alpha\rangle E^*$.

Case 3. α is a move whose effect is making a catch-up switch in some active surface occurrence of a $\triangledown$-subformula. According to clause 6 of Lemma 6.7, $\langle\Omega,\bot\alpha\rangle$ remains an E-manageable legal position of E^*. In this case, as in the previous two cases, $\mathcal{S}$ calls itself on $\langle\Omega,\bot\alpha\rangle E^*$.

Case 4. α is a move whose effect is making a move γ in some active surface occurrence of a non-widowed hybrid atom. Let β be the move whose effect is making the same move γ within the other active surface occurrence of the same hybrid atom. According to clause 7 of Lemma 6.7, $\langle\Omega,\bot\alpha,\top\beta\rangle$ remains an E-manageable legal position of E^*. In this case, $\mathcal{S}$ makes the move β and calls itself on $\langle\Omega,\bot\alpha,\top\beta\rangle E^*$.

Case 5: α is a move whose effect is a choice of the ith component in an active surface occurrence of a subformula $G_1 \sqcap \ldots \sqcap G_n$. Then $\mathcal{S}$ switches to its winning strategy for $\langle\Omega\rangle H^*$, where H is the result of replacing the above subformula by G_i in E. Clause 8 of Lemma 6.7 guarantees that $\langle\Omega\rangle H^*$ is indeed the game to which $\langle\Omega\rangle E^*$ has evolved and that Ω is an H-manageable legal position of H^*, so that, by the induction hypothesis, $\mathcal{S}$ knows how to win $\langle\Omega\rangle H^*$.

Case 6: α signifies a (leading) switch move within an active surface occurrence of a subformula

$$G_0 \triangle \ldots \triangle \underline{G_m} \triangle G_{m+1} \triangle \ldots \triangle G_n.$$

Then $\mathcal{S}$ makes the same move α (signifying making a catch-up switch within the same subformula), and calls itself on $\langle\Omega,\bot\alpha,\top\alpha\rangle H^*$, where H is the result of replacing the above subformula by

$$G_0 \triangle \ldots \triangle G_m \triangle \underline{G_{m+1}} \triangle \ldots \triangle G_n.$$

Clause of 9 Lemma 6.7 guarantees that $\langle\Omega,\bot\alpha,\top\alpha\rangle$ is an H-manageable legal position of H^*, so that, by the induction hypothesis, $\mathcal{S}$ will win $\langle\Omega,\bot\alpha,\top\alpha\rangle H^*$.

Looking back at $\mathcal{S}$'s strategy, the value of E keeps changing from conclusion to a premise, starting from the original formula F. It is therefore obvious that such a value should stabilize at some E_{final} and never change afterwards. It is also obvious that this E_{final} should be derived by Wait$^\circ$, or else it would further change. For the same reason, while $\mathcal{S}$ acts following the prescriptions for the case of Wait$^\circ$ with $E = E_{final}$, Cases 5 and 6 never occur. But in all other four cases, as well as in the case when $\bot$ does not make any moves, (the perhaps continuously updated) Ω remains E_{final}-manageable, and—according to Lemma 6.5—Ω will remain so even if it grows infinite. Also, as a conclusion of Wait, E_{final} is stable. This, by Lemma 6.10, implies that the overall game will be won by $\mathcal{S}$.

To officially complete our proof of the lemma, it remains to note that $\mathcal{S}$, of course, can be constructed effectively from a given **CL9**$^\circ$-proof of F. □

8. Preliminaries for the completeness proof

8.1. Machines against machines

This subsection borrows a discussion from [6], providing certain background information necessary for our completeness proof but missing in [16], the only external source on computability logic on which the present paper was promised to rely.

For a run Γ and a computation branch B of an HPM or EPM, we say that B **cospells** Γ iff B spells $\neg\Gamma$ (Γ with all labels reversed) in the sense of Section 6 of [16]. Intuitively, when a machine $\mathcal{M}$ plays as $\bot$ (rather than $\top$), then the run that is generated by a given computation branch B of $\mathcal{M}$ is the run cospelled (rather than spelled) by B, for the moves that $\mathcal{M}$ makes get the label $\bot$, and the moves that its adversary makes get the label $\top$.

We say that an EPM $\mathcal{E}$ is **fair** iff, for every valuation e, every e-computation branch of $\mathcal{E}$ is fair in the sense of Section 6 of [16].

Lemma 8.1. *Assume $\mathcal{E}$ is a fair EPM, $\mathcal{H}$ is any HPM, and e is any valuation. There are a uniquely defined e-computation branch $B_\mathcal{E}$ of $\mathcal{E}$ and a uniquely defined e-computation branch $B_\mathcal{H}$ of $\mathcal{H}$—which we, respectively, call* **the** $(\mathcal{E},e,\mathcal{H})$**-branch** *and* **the** $(\mathcal{H},e,\mathcal{E})$**-branch**—*such that the run spelled by $B_\mathcal{H}$, called* **the** $\mathcal{H}$ **vs**. $\mathcal{E}$ **run on** e, *is the run cospelled by $B_\mathcal{E}$.*

When $\mathcal{H},\mathcal{E},e$ are as above, Γ is the $\mathcal{H}$ vs. $\mathcal{E}$ run on e and A is a game with $\mathbf{Wn}_e^A\langle\Gamma\rangle = \top$ (resp. $\mathbf{Wn}_e^A\langle\Gamma\rangle = \bot$), we say that $\mathcal{H}$ **wins** (resp. **loses**) A **against** $\mathcal{E}$ **on** e.

A strict proof of the above lemma can be found in [5] (Lemma 20.4), and we will not reproduce the formal proof here. Instead, the following intuitive explanation should suffice.

Proof idea. Assume $\mathcal{H}$, $\mathcal{E}$, e are as in Lemma 8.1. The play that we are going to describe is the unique play generated when the two machines play against each other, with $\mathcal{H}$ in the role of $\top$, $\mathcal{E}$ in the role of $\bot$, and e spelled on the valuation tapes of both machines. We can visualize this play as follows. Most of the time during the process $\mathcal{H}$ remains inactive (sleeping); it is woken up only when $\mathcal{E}$ enters a permission state, on which event $\mathcal{H}$ makes a (one single) transition to its next computation step—that may or may not result in making a move—and goes back into a sleep that will continue until $\mathcal{E}$ enters a permission state again, and so on. From $\mathcal{E}$'s perspective, $\mathcal{H}$ acts as a patient adversary who makes one or zero move only when granted permission, just as the EPM-model assumes. And from $\mathcal{H}$'s perspective, which, like a person in a coma, has no sense of time during its sleep and hence can think that the wake-up events that it calls the beginning of a clock cycle happen at a constant rate, $\mathcal{E}$ acts as an adversary who can make any finite number of moves during a clock cycle (i.e., while $\mathcal{H}$ was sleeping), just as the HPM-model assumes. This scenario uniquely determines an e-computation branch $B_{\mathcal{E}}$ of $\mathcal{E}$ that we call the $(\mathcal{E},e,\mathcal{H})$-branch, and an e-computation branch $B_{\mathcal{H}}$ of $\mathcal{H}$ that we call the $(\mathcal{H},e,\mathcal{E})$-branch. What we call the $\mathcal{H}$ vs. $\mathcal{E}$ run on e is the run generated in this play. In particular—since we let $\mathcal{H}$ play in the role of $\top$—this is the run spelled by $B_{\mathcal{H}}$. $\mathcal{E}$, who plays in the role of $\bot$, sees the same run, only it sees the labels of the moves of that run in negative colors. That is, $B_{\mathcal{E}}$ cospells rather than spells that run. This is exactly what Lemma 8.1 asserts. □

8.2. Logics **CL10**, **CL10**$^\circ$ *and* $\overline{\textbf{CL10}^\circ}$

Our proof of the completeness part of Theorem 5.7 employs the conservative, elementary-base fragment **CL10** of **CL9**, obtained by restricting the language of the latter to elementary-base formulas—we refer to formulas of this restricted language as **CL10-formulas**—and (correspondingly) deleting the rule of Match. Logic **CL1**, historically the first system for computability logic proven (in [6]) to be sound and complete, is a $\triangle$,$\triangledown$-free counterpart of **CL10**. Of course, **CL10** inherits soundness from **CL9**. In this section we are going to prove the completeness of **CL10**.

Our completeness proof for **CL10**, in turn, employs the modification **CL10**$^\circ$ of **CL10**. **CL10**$^\circ$**-formulas** are nothing but elementary-base **CL9**$^\circ$-formulas, i.e., **CL9**$^\circ$-formulas that contain no general or hybrid atoms. Thus, the only difference between **CL10**-formulas and **CL10**$^\circ$-formulas is that, in the latter one of the components of each sequential subformula is underlined.

Logic **CL10**$^\circ$ is a conservative fragment of logic **CL9**$^\circ$ in the same sense as **CL10** is a fragment of **CL9**. Namely, **CL10**$^\circ$ is obtained from **CL9**$^\circ$ by restricting its language to elementary-base formulas, and deleting the rule of Match$^\circ$. Here we (re)produce the rules of **CL10**$^\circ$ for the convenience of later references:

Definition 8.2. The inference rules of **CL10**$^\circ$ are:

Wait$^\circ$: $\vec{H} \vdash F$, where F is stable and $\vec{H}$ is the smallest set of formulas satisfying the following two conditions:

- whenever F has an active surface occurrence of a subformula $G_1 \sqcap \ldots \sqcap G_n$, for each $i \in \{1,\ldots,n\}$, $\vec{H}$ contains the result of replacing that occurrence in F by G_i;
- whenever F has an active surface occurrence of a subformula

  $G_0 \triangle \ldots \triangle \underline{G_m} \triangle G_{m+1} \triangle \ldots \triangle G_n$,

  $\vec{H}$ contains the result of replacing that occurrence in F by

  $G_0 \triangle \ldots \triangle G_m \triangle \underline{G_{m+1}} \triangle \ldots \triangle G_n$.

Choose$^\circ$: $H \vdash F$, where H is the result of replacing in F an active surface occurrence of a subformula $G_1 \sqcup \ldots \sqcup G_n$ by G_i for some $i \in \{1,\ldots,n\}$.

Switch$^\circ$: $H \vdash F$, where H is the result of replacing in F an active surface occurrence of a subformula

$G_0 \triangledown \ldots \triangledown \underline{G_m} \triangledown G_{m+1} \triangledown \ldots \triangledown G_n$

by

$G_0 \triangledown \ldots \triangledown G_m \triangledown \underline{G_{m+1}} \triangledown \ldots \triangledown G_n$.

The following lemma establishes equivalence between **CL10** and **CL10**$^\circ$.

Lemma 8.3. *For any formula* G, **CL10**$^\circ \vdash G$ *iff* **CL10** $\vdash G$.

Proof. This lemma is pretty straightforward: a **CL10**$^\circ$-proof of G turns into a **CL10**-proof of G after replacing, in each formula of the former, every subformula $E_1 \triangle \ldots \triangle \underline{E_m} \triangle \ldots \triangle E_n$ by $E_m \triangle \ldots \triangle E_n$ and every subformula $E_1 \triangledown \ldots \triangledown \underline{E_m} \triangledown \ldots \triangledown E_n$ by $E_m \triangledown \ldots \triangledown E_n$. And vice versa: a **CL10**-proof of G turns into a **CL10**$^\circ$-proof of G after underlining the head of each sequential subformula of G. Then, in the bottom up view of the proof, every time Switch is used, it should simply move the underline

to the next sequential component instead of deleting the head. Similarly for the premises of Wait that are associated with sequential subformulas. □

Next, we define Logic $\overline{\mathbf{CL10}^\circ}$ which is a "dual" of $\mathbf{CL10}^\circ$.

Definition 8.4. The language of $\overline{\mathbf{CL10}^\circ}$ is the same as that of $\mathbf{CL10}^\circ$, and the rules of inference are:

$\overline{\text{Wait}^\circ}$: $\vec{H} \vdash F$, where F is instable and $\vec{H}$ is the smallest set of formulas satisfying the following two conditions:

- whenever F has an active surface occurrence of a subformula $G_1 \sqcup \ldots \sqcup G_n$, for each $i \in \{1,\ldots,n\}$, $\vec{H}$ contains the result of replacing that occurrence in F by G_i;
- whenever F has an active surface occurrence of a subformula

  $G_0 \triangledown \ldots \triangledown \underline{G_m} \triangledown G_{m+1} \triangledown \ldots \triangledown G_n$,

  $\vec{H}$ contains the result of replacing that occurrence in F by

  $G_0 \triangledown \ldots \triangledown G_m \triangledown \underline{G_{m+1}} \triangledown \ldots \triangledown G_n$.

$\overline{\text{Choose}^\circ}$: $H \vdash F$, where H is the result of replacing in F an active surface occurrence of a subformula $G_1 \sqcap \ldots \sqcap G_n$ by G_i for some $i \in \{1,\ldots,n\}$.

$\overline{\text{Switch}^\circ}$: $H \vdash F$, where H is the result of replacing in F an active surface occurrence of a subformula

$G_0 \triangle \ldots \triangle \underline{G_m} \triangle G_{m+1} \triangle \ldots \triangle G_n$

by

$G_0 \triangle \ldots \triangle G_m \triangle \underline{G_{m+1}} \triangle \ldots \triangle G_n$.

Lemma 8.5. $\mathbf{CL10}^\circ \not\vdash F$ *iff* $\overline{\mathbf{CL10}^\circ} \vdash F$ (*any* $\mathbf{CL10}^\circ$*-formula* F).

Proof. We prove this lemma by induction on the complexity of F. It would be sufficient to verify the 'only if' part, as the 'if' part (which we do not need anyway) can be handled in a fully symmetric way. So, assume $\mathbf{CL10}^\circ \not\vdash F$ and let us see that then $\overline{\mathbf{CL10}^\circ} \vdash F$. There are two cases to consider:

Case 1: F is stable. Then there must be a $\mathbf{CL10}^\circ$-unprovable formula H satisfying one of the following two conditions, for otherwise F would be $\mathbf{CL10}^\circ$-derivable by Wait$^\circ$:

- H is the result of replacing in F an active surface occurrence of a subformula $G_1 \sqcap \ldots \sqcap G_n$ by G_i for some $i \in \{1,\ldots,n\}$.
- H is the result of replacing in F an active surface occurrence of a subformula

  $G_0 \triangle \ldots \triangle \underline{G_m} \triangle G_{m+1} \triangle \ldots \triangle G_n$

  by

  $G_0 \triangle \ldots \triangle G_m \triangle \underline{G_{m+1}} \triangle \ldots \triangle G_n$.

In either case, by the induction hypothesis, $\overline{\mathbf{CL10}^\circ} \vdash H$, whence, by $\overline{\text{Choose}^\circ}$ (if the first condition is satisfied) or $\overline{\text{Switch}^\circ}$ (if the second condition is satisfied), $\overline{\mathbf{CL10}^\circ} \vdash F$.

Case 2: F is instable. Let $\vec{H}$ be the smallest set of formulas such that the following two conditions are satisfied:

- whenever F has an active surface occurrence of a subformula $G_1 \sqcup \ldots \sqcup G_n$, for each $i \in \{1,\ldots,n\}$, $\vec{H}$ contains the result of replacing that occurrence in F by G_i;
- whenever F has an active surface occurrence of a subformula

  $G_0 \triangledown \ldots \triangledown \underline{G_m} \triangledown G_{m+1} \triangledown \ldots \triangledown G_n$,

  $\vec{H}$ contains the result of replacing that occurrence in F by

  $G_0 \triangledown \ldots \triangledown G_m \triangledown \underline{G_{m+1}} \triangledown \ldots \triangledown G_n$.

None of the elements of $\vec{H}$ is provable in $\mathbf{CL10}^\circ$, for otherwise F would also be derivable in $\mathbf{CL10}^\circ$ by Choose$^\circ$ or Switch$^\circ$. Therefore, by the induction hypothesis, each element of $\vec{H}$ is $\overline{\mathbf{CL10}^\circ}$-provable, whence, by $\overline{\text{Wait}^\circ}$, we have $\overline{\mathbf{CL10}^\circ} \vdash F$. □

8.3. The completeness of **CL10**

Lemma 8.6. *If* $\mathbf{CL10} \not\vdash F$, *then* F *is not valid* (*any* $\mathbf{CL10}$*-formula* F).

In particular, if $\mathbf{CL10} \not\vdash F$, *then* F^* *is not computable for some interpretation* * *that interprets all atoms as finitary predicates of complexity* Δ_2.

Proof. Assume $\mathbf{CL10} \not\vdash F$, which, by Lemmas 8.3 and 8.5, means that $\overline{\mathbf{CL10}^{\circ}} \vdash F$. Fix a $\overline{\mathbf{CL10}^{\circ}}$-proof of F. From such a proof, we can extract an environment's interpretation- and valuation-independent EPM-counterstrategy $\mathcal{C}$ for F in a way fully symmetric to the way we extracted the machine's strategy $\mathcal{S}$ from a $\mathbf{CL9}^{\circ}$-proof in Section 7. $\mathcal{C}$ is a *counterstrategy* in the sense that $\mathcal{C}$ plays in the role of $\bot$ rather than $\top$.[14] In fact, $\mathcal{C}$ is much simpler than $\mathcal{S}$, because in the present case we only deal with elementary-base formulas. Since the work of $\mathcal{C}$ depends neither on * nor on e, in our description of it we can safely omit these parameters, and write E instead of $e[E^*]$.

The strategy that $\mathcal{C}$ follows is a recursive one which, just like its counterpart from Section 7, at every step deals with $\langle\Omega\rangle E$, where E is some formula from the $\overline{\mathbf{CL10}^{\circ}}$-proof of F, and Ω is a certain legal position of E. Initially $E = F$ and $\Omega = \langle\rangle$. How $\mathcal{C}$ acts on $\langle\Omega\rangle E$ depends on by which of the three rules E is derived in $\overline{\mathbf{CL10}^{\circ}}$.

If E is derived by $\overline{\text{Choose}^{\circ}}$ from H as described in Definition 8.4, $\mathcal{C}$ makes the move α whose effect is choosing G_i in the $G_1 \sqcap \ldots \sqcap G_n$ subformula of E. Then it updates E to H (without changing Ω), and calls (repeats) itself.

If E is derived by $\overline{\text{Switch}^{\circ}}$ from H as described in Definition 8.4, then $\mathcal{C}$ makes the move α whose effect is making a switch in the $G_0 \triangle \ldots \triangle \underline{G_m} \triangle G_{m+1} \triangle \ldots \triangle G_n$ component, updates E to H (without changing Ω), and calls itself.

Finally, suppose E is derived by $\overline{\text{Wait}^{\circ}}$. $\mathcal{C}$ keeps granting permission. Now and then the adversary may be making moves in abandoned components, or catch-up switch moves within sequential subformulas. To such moves $\mathcal{C}$ reacts by adding them to its internal record of Ω (updating Ω), but otherwise $\mathcal{C}$ does not move. However, what will typically happen during this stage (except one—the last—case) is that sooner or later the adversary makes a legal move α[15] that causes $\mathcal{C}$ to update E to one of its premises. In particular, one of the following will be the case:

Case 1: α is a move signifying a choice of the ith component in an active surface occurrence of a subformula $G_1 \sqcup \ldots \sqcup G_n$. Then $\mathcal{C}$ updates E to the result of replacing in E the above subformula by G_i (without changing Ω), and calls itself.

Case 2: α is a move signifying a switch in an active surface occurrence of a subformula

$$G_0 \triangledown \ldots \triangledown \underline{G_m} \triangledown G_{m+1} \triangledown \ldots \triangledown G_n.$$

Then $\mathcal{C}$ makes the same move α, updates E to the result of replacing in it the above subformula by $G_0 \triangledown \ldots \triangledown G_m \triangledown \underline{G_{m+1}} \triangledown \ldots \triangledown G_n$, updates Ω to $\langle\Omega, \bot\alpha, \top\alpha\rangle$, and calls itself.

As we can see from the above description, the value of E starts with F and, moving up along one of the branches of the $\overline{\mathbf{CL10}^{\circ}}$-proof of F, eventually stabilizes at $E = E_{\mathit{final}}$ for one of the instable formulas of the proof (E_{final} is instable because it is the conclusion of $\overline{\text{Wait}^{\circ}}$). Call such a formula E_{final} the **limit formula** of the given play. Of course, $\mathcal{C}$ is a fair EPM because it will grant permission infinitely many times after reaching the limit formula.

For reasons fully symmetric to those that we relied upon in Section 7, at each step of the work of $\mathcal{C}$, $\langle\Omega\rangle E$ can be seen to be exactly the game to which the original game F has been brought down in the play. So, where Ω_{final} is the final value of Ω, $\mathcal{C}$ will be the winner in the overall play over F iff Ω_{final} is a $\bot$-won run of E_{final}. More precisely, for any given valuation e and interpretation * (the parameters that we have been suppressing so far), the overall play over $e[F^*]$ is won by $\mathcal{C}$ iff Ω_{final} is a $\bot$-won run of $e[E^*_{\mathit{final}}]$. Next, taking into account that Ω_{final} only contains (the meaningless) moves in abandoned components and catch-up switches, with some thought one can see that Ω_{final} is a $\bot$-won run of $e[E^*_{\mathit{final}}]$ if and only if $\|E_{\mathit{final}}\|^*$ is false at e.

Of course, different $\top$'s strategies (HPMs) $\mathcal{H}$ and different valuations e may yield different runs and hence induce different limit formulas. So, our goal now is to select an interpretation * such that, for any HPM $\mathcal{H}$, there is a valuation e at which $\|E_{\mathit{final}}\|^*$ is false, where E_{final} is the limit formula of the play of $\mathcal{H}$ against $\mathcal{C}$ on valuation e. This would mean that no HPM can win F^* against $\mathcal{C}$.

Let us fix some standard way of describing HPMs, and let

$$\mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3, \ldots$$

be the list of all HPMs arranged according to the lexicographic order of their descriptions, so that each constant c can be considered the code of $\mathcal{H}_c$. Next, we fix a variable x and agree that, for each constant c,

$$e_c$$

is the valuation with $e_c(x) = c$ (and, say, $e_c(y) = 1$ for any other variable $y \neq x$). Further,

$$L_c$$

will denote the limit formula of the game over F between $\mathcal{H}_c$, in the role of $\top$, and our $\mathcal{C}$, in the role of $\bot$, on valuation e_c. In more precise terms, L_c is the limit formula induced by the $\mathcal{H}_c$ vs. $\mathcal{C}$ run on e_c (remember Lemma 8.1).

Finally, let

$$G_1, \ldots, G_k$$

[14] If we want to see $\mathcal{C}$ as a strategy in the ordinary sense, then it is a strategy for $\neg F$.

[15] And if α is illegal, then $\mathcal{C}$'s job is done.

be all (instable) formulas from the $\overline{\mathbf{CL10}^{\circ}}$-proof of F that are obtained by $\overline{\mathrm{Wait}^{\circ}}$. For each such G_i, we fix a classical model (true/false assignment for atoms) M_i such that

M_i makes the elementarization of G_i false.

And, for each $i \in \{1,\ldots,k\}$, we define the predicate T_i by

T_i is true at a valuation e iff $L_{e(x)} = G_i$.

Now we define the interpretation * by stipulating that, for each atom p,

$p^* = \vee\{T_i \mid 1 \le i \le k, p \text{ is true in } M_i\}$

($\vee\mathcal{F}$ means the $\vee$-disjunction of all elements of $\mathcal{F}$, understood as $\bot$ when the set $\mathcal{F}$ is empty.)

Consider an arbitrary $c \in \{1,2,\ldots\}$. As noted earlier, we must have $L_c = G_j$ for one of the $j \in \{1,\ldots,k\}$. Fix this j. Observe that, at valuation e_c, T_j is true and all other T_i ($i \ne j$, $1 \le i \le k$) are false. With this fact in mind, it is easy to see that, for every atom p, p is true in M_j iff the predicate p^* is true at e_c. This obviously extends from atoms to their $\neg, \wedge, \vee$-combinations, so that $\|G_j\|$ is true in M_j iff the predicate $\|G_j\|^*$ is true at e_c. And, by our choice of the models M_i, the formula $\|G_j\|$, i.e., $\|L_c\|$, is false in M_j. Consequently, $\|L_c\|^*$ is false at e_c. But L_c is the limit formula of the play between $\mathcal{H}$ and $\mathcal{C}$ on e_c and, as we noted earlier, the fact that $\|L_c\|^*$ is false at e_c implies that $\mathcal{H}$ loses F^* against $\mathcal{C}$ on valuation e_c.

Thus, no $\mathcal{H}_c$ computes F^*, meaning that F^* is not computable, because every HPM is $\mathcal{H}_c$ for some c. Note also that, as promised in the lemma, the predicate p^* (any atom p) is finitary as only the value assigned to x matters. To officially complete the present proof, it remains to show that

the complexity of p^* is Δ_2 (any atom p). (17)

Remember that an arithmetical predicate $A(c)$ (with c here seen as a variable) is said to have complexity Σ_2 iff it can be written as $\exists x \forall y B(c,x,y)$ for some decidable predicate $B(c,x,y)$; and $A(c)$ is of complexity Δ_2 iff both $A(c)$ and $\neg A(c)$ are of complexity Σ_2.

We defined p^* as a disjunction of some T_i, that we now think of as unary arithmetical predicates and write as $T_i(c)$. Disjunction is known to preserve Δ_2—as well as Σ_2 — complexity, so, in order to verify (17), it would be sufficient to show that each $T_i(c)$ ($1 \le i \le k$) is of complexity Δ_2. Looking at the meaning of $T_i(c)$, this predicate asserts nothing but that the value of $\mathcal{C}$'s internal record of E in the process of playing the $(\mathcal{C},e_c,\mathcal{H}_c)$-branch (see Lemma 8.1)—call this branch B_c—will stabilize at G_i. So, $T_i(c)$ can be written as $\exists x \forall y (y \ge x \to K_i(c,y))$, where $K_i(c,y)$ means "the value of record E at the yth computation step of branch B_c is G_i". Furthermore, we know that the value of E should indeed stabilize at one of the instable formulas $G_1,\ldots,G_k$ of the proof. Hence, $\neg T_i(c)$ is equivalent to $\vee\{T_j(c) \mid 1 \le j \le k, j \ne i\}$. Consequently, in order to show that each $T_i(c)$ is of complexity Δ_2, it would suffice to show that each $T_i(c)$ is of complexity Σ_2. For the latter, in turn, verifying that $K_i(c,y)$ is a decidable predicate would be sufficient. But $K_i(c,y)$ is indeed decidable. A decision procedure for it first constructs the machine $\mathcal{H}_c$ from number c. Then it lets this machine play against $\mathcal{C}$ on valuation e_c as described in the proof idea for Lemma 8.1. In particular, it traces, in parallel, how the configurations of the two machines evolve up to the yth computation step of $\mathcal{C}$, i.e., its yth configuration. Then the procedure looks at the value of record E in that configuration, and says "yes" or "no" depending on whether the latter is G_i or not. □

9. The completeness of CL9

Lemma 9.1. *If* **CL9** $\not\vdash F$, *then F is not valid* (*any* **CL9**-*formula F*).

Moreover, if **CL9** $\not\vdash F$, *then F^* is not computable for some interpretation * that interprets all elementary atoms of F as finitary predicates of arithmetical complexity Δ_2, and interprets all general atoms of F as problems of the form $(A^1_1 \sqcup \ldots \sqcup A^1_m) \sqcap \ldots \sqcap (A^m_1 \sqcup \ldots \sqcup A^m_m)$, where each A^j_i is a finitary predicate of arithmetical complexity Δ_2.*

Proof idea. We are going to show that if **CL9** $\not\vdash F$, then there is a **CL10**-formula $\ulcorner F \urcorner$ of the same form as F that is not provable in **CL10**. Precisely, "the same form as F" here means that $\ulcorner F \urcorner$ is the result of rewriting/expanding in F every general atom P as a certain elementary-base formula $\check{P}^{\sqcap}_{\sqcup}$. This, in view of the already known completeness of **CL10**, immediately yields non-validity for F. As it turns out, the above formulas $\check{P}^{\sqcap}_{\sqcup}$, that we call *molecules*, can be chosen to be as simple as sufficiently long $\sqcap$-conjunctions of sufficiently long $\sqcup$-disjunctions of arbitrary "neutral" (not occurring in F and pairwise distinct) elementary atoms, with the "sufficient length" of those conjunctions/disjunctions being bounded by the number of occurrences of general atoms in F.

Intuitively, the reason why **CL10** $\not\vdash \ulcorner F \urcorner$, i.e., why $\top$ cannot win (the game represented by) $\ulcorner F \urcorner$, is that a smart environment may start choosing different conjuncts/disjuncts in different occurrences of $\check{P}^{\sqcap}_{\sqcup}$. The best that $\top$ can do in such a play is to match any given positive or negative occurrence of $\check{P}^{\sqcap}_{\sqcup}$ with one (but not more!) negative or positive occurrence of the same subgame—match in the sense that mimic environment's moves in order to keep the subgames/subformulas at the two occurrences identical. Yet, this is insufficient for $\top$ to achieve a guaranteed success. This is so because $\top$'s matching decisions for $\ulcorner F \urcorner$ could be modeled by appropriate applications of the rule of Match in an attempted **CL9**-proof for F, and so can be—through the rules of Wait, Choose and Switch — either player's decisions required by choice and sequential connectives in

the non-molecule parts of $\ulcorner F\urcorner$. A winning strategy (**CL10**-proof) for $\ulcorner F\urcorner$ would then translate into a **CL9**-proof for F, which, however, does not exist.

Proof. Fix a **CL9**-formula F. Let $\mathcal{P}$ be the set of all general atoms occurring in F. Let us fix m as the total number of occurrences of such atoms in F;[16] if there are fewer than 2 of such occurrences, then we take $m = 2$.

For the rest of this section, let us agree that

a,b always range over $\{1,\ldots,m\}$.

For each $P \in \mathcal{P}$ and each a,b, let us fix an elementary atom

- $\check{P}^a_b$

not occurring in F. We assume that $\check{P}^a_b \neq \check{Q}^c_d$ as long as either $P \neq Q$ or $a \neq c$ or $b \neq d$. Note that the $\check{P}^a_b$ are elementary atoms despite our "tradition" according to which the capital letters $P,Q,\ldots$ stand for general atoms.

Next, for each $P \in \mathcal{P}$ and each a, we define

- $\check{P}^a_{\sqcup} = \check{P}^a_1 \sqcup \ldots \sqcup \check{P}^a_m$.

Finally, for each $P \in \mathcal{P}$, we define

- $\check{P}^{\sqcap}_{\sqcup} = \check{P}^1_{\sqcup} \sqcap \ldots \sqcap \check{P}^m_{\sqcup}$, i.e., $\check{P}^{\sqcap}_{\sqcup} = \left(\check{P}^1_1 \sqcup \ldots \sqcup \check{P}^1_m\right) \sqcap \ldots \sqcap \left(\check{P}^m_1 \sqcup \ldots \sqcup \check{P}^m_m\right)$.

We refer to the above formulas $\check{P}^a_b$, $\check{P}^a_{\sqcup}$ and $\check{P}^{\sqcap}_{\sqcup}$ as **molecules**, in particular, P**-based molecules**. To differentiate between the three sorts of molecules, we call the molecules of the type $\check{P}^a_b$ **small**, call the molecules of the type $\check{P}^a_{\sqcup}$ **medium**, and call the molecules of the type $\check{P}^{\sqcap}_{\sqcup}$ **large**. Thus, where k is the cardinality of $\mathcal{P}$, altogether there are k large molecules, $k \times m$ medium molecules and $k \times m \times m$ small molecules.

For simplicity, for the rest of this section we assume/pretend that the languages of **CL9** and **CL10** have no non-logical atoms other than those occurring in F plus the atoms $\check{P}^b_a$ ($P \in \mathcal{P}$, $a,b \in \{1,\ldots,m\}$). This way the scope of the term "formula" is correspondingly redefined.

An occurrence of a molecule M in a formula can be **positive** or **negative**. While a positive occurrence literally means M, a negative occurrence looks like $\neg M$, which—unless M is a small molecule—should be considered a standard abbreviation. For example, a negative occurrence of the medium molecule $\check{P}^a_1 \sqcup \ldots \sqcup \check{P}^a_m$ is nothing but an ("ordinary", positive) occurrence of $\neg\check{P}^a_1 \sqcap \ldots \sqcap \neg\check{P}^a_m$. One should be especially careful when applying the terms "positive occurrence" and "negative occurrence" to small molecules, as here the meaning of our terminology somewhat diverges from its earlier-used meaning for atoms. Specifically, a positive occurrence of a small molecule $\check{P}^a_b$ means—as expected—an occurrence that comes without $\neg$ in the formula. As for a negative occurrence of the *molecule* $\check{P}^a_b$ (as opposed to the *atom* $\check{P}^a_b$), it means an occurrence of the subformula $\neg\check{P}^a_b$ rather than just the $\check{P}^a_b$ part of it under $\neg$. So, for example, the result of replacing the negative occurrence of $\check{P}^a_b$ by Q in the formula $E \vee \neg\check{P}^a_b$ is the formula $E \vee Q$ rather than $E \vee \neg Q$, as $\neg$ was a part of what we call a "negative occurrence of $\check{P}^a_b$".

Let us say that a (positive or negative) occurrence of a molecule in a given **CL10**-formula is **independent** iff it is not a part of another ("larger") molecule. For example, the negative occurrence of the medium molecule $\check{P}^1_1 \sqcup \ldots \sqcup \check{P}^1_m$ in the following formula is independent while its positive occurrence is not:

$$\left(\neg\check{P}^1_1 \sqcap \ldots \sqcap \neg\check{P}^1_m\right) \vee \left(\left(\check{P}^1_1 \sqcup \ldots \sqcup \check{P}^1_m\right) \sqcap \ldots \sqcap \left(\check{P}^m_1 \sqcup \ldots \sqcup \check{P}^m_m\right)\right).$$

Of course, surface occurrences of molecules are always independent, and so are any—surface or non-surface—occurrences of large molecules.

We say that a **CL10**-formula E is **good** iff the following conditions are satisfied:

Condition (i): E contains at most m independent occurrences of molecules.
Condition (ii): Only large molecules (may) have independent non-surface occurrences in E.
Condition (iii): Each small molecule has at most one positive and at most one negative independent occurrence in E.
Condition (iv): For each medium molecule $\check{P}^a_{\sqcup}$, E has at most one positive independent occurrence of $\check{P}^a_{\sqcup}$, and when E has such an occurrence, then for no b does E have a positive independent occurrence of the small molecule $\check{P}^a_b$.

Let E be a **CL10**-formula. By an **isolated** small molecule of E (or E**-isolated** small molecule, or a small molecule **isolated in** E) we will mean a small molecule that has exactly one independent occurrence in E. We will say that such a molecule is **positive** or **negative** depending on whether its independent occurrence in E is positive or negative.

Next, the **floorification** of E, denoted

$$\lfloor E \rfloor,$$

[16] In fact, a much smaller m would be sufficient for our purposes. E.g., m can be chosen to be such that no given general atom has more than m occurrences in F. But why try to economize?

is the result of replacing in E every positive (resp. negative) independent occurrence of every P-based (each $P \in \mathcal{P}$) large, medium and E-isolated small molecule[17] by the general literal P (resp. $\neg P$).

Claim 1. *For any good* **CL10***-formula* E*, if* $\mathbf{CL10} \vdash E$*, then* $\mathbf{CL9} \vdash \lfloor E \rfloor$.

To prove this claim, assume E is a good **CL10**-formula, and $\mathbf{CL10} \vdash E$. By induction on the length of the **CL10**-proof of E, we want to show that $\mathbf{CL9} \vdash \lfloor E \rfloor$. We need to consider the following three cases, depending on which of the three rules of **CL10** was used (last) to derive E.

Case 1: E is derived by Wait. Let us fix the set $\vec{H}$ of premises of E. Each formula $H \in \vec{H}$ is provable in **CL10**. Hence, by the induction hypothesis, we have:

For any $H \in \vec{H}$*, if* H *is good, then* $\mathbf{CL9} \vdash \lfloor H \rfloor$. (18)

We consider the following three subcases. The first two subcases are not mutually exclusive, and either one can be chosen when both of them apply. Specifically, Subcase 1.1 (resp. 1.2) is about when E has a positive (resp. negative) surface occurrence of a large (resp. medium) molecule. Then, as we are going to see, replacing that molecule by a "safe" $\sqcap$-conjunct of it, corresponding to a smart environment's possible move, yields a good formula H from $\vec{H}$ such that $\lfloor E \rfloor = \lfloor H \rfloor$. This, by (18), automatically means the **CL9**-provability of $\lfloor E \rfloor$. The remaining Subcase 1.3 is about when all surface occurrences of large (resp. medium) molecules in E are negative (resp. positive). This will be shown to imply that $\lfloor E \rfloor$ follows from the floorifications of some elements of $\vec{H}$ by Wait for "the same reasons as" E follows from $\vec{H}$.

Subcase 1.1: E has a positive surface occurrence of a large molecule $\check{P}^{\sqcap}_{\sqcup}$, i.e., an occurrence of

$$\check{P}^1_{\sqcup} \sqcap \ldots \sqcap \check{P}^m_{\sqcup}.$$

Pick any $a \in \{1, \ldots, m\}$ such that neither the medium molecule $\check{P}^a_{\sqcup}$ nor any small molecule $\check{P}^a_b$ (whatever b) have independent occurrences in E. Such an a exists, for otherwise we would have at least $m+1$ independent occurrences of molecules in E (including the occurrence of $\check{P}^{\sqcap}_{\sqcup}$), which violates Condition (i) of the definition of "goodness". Let H be the result of replacing in E the above occurrence of $\check{P}^{\sqcap}_{\sqcup}$ by $\check{P}^a_{\sqcup}$. Clearly $H \in \vec{H}$. Observe that when transferring from E to H, we just "downsize" $\check{P}^{\sqcap}_{\sqcup}$ and otherwise do not create any additional independent occurrences of molecules, so Condition (i) continues to be satisfied for H. Neither do we introduce any new non-surface occurrences of molecules or any new independent occurrences of small molecules, so Conditions (ii) and (iii) also continue to hold for H. And our choice of a obviously guarantees that so does Condition (iv). To summarize, H is good. Therefore, by (18), $\mathbf{CL9} \vdash \lfloor H \rfloor$. Finally, note that, when floorifying a given formula, both $\check{P}^{\sqcap}_{\sqcup}$ and $\check{P}^a_{\sqcup}$ get replaced by the same atom P; and, as the only difference between E and H is that H has $\check{P}^a_{\sqcup}$ where E has $\check{P}^{\sqcap}_{\sqcup}$, obviously $\lfloor H \rfloor = \lfloor E \rfloor$. Thus, $\mathbf{CL9} \vdash \lfloor E \rfloor$.

Subcase 1.2: E has a negative surface occurrence of a medium molecule $\check{P}^a_{\sqcup}$—that is, an occurrence of

$$\neg\check{P}^1_a \sqcap \ldots \sqcap \neg\check{P}^m_a.$$

Pick any b such that E does not have an independent occurrence of $\check{P}^a_b$. Again, in view of Condition (i), such a b exists. Let H be the result of replacing in E the above occurrence of $\neg\check{P}^1_a \sqcap \ldots \sqcap \neg\check{P}^m_a$ by $\neg\check{P}^a_b$. Certainly $H \in \vec{H}$. Conditions (i) and (ii) continue to hold for H for the same reasons as in Subcase 1.1. In view of our choice of b, Condition (iii) is also inherited by H from E. And so is Condition (iv), because H has the same positive occurrences of (the same) molecules as E does. Thus, H is good. Therefore, by (18), $\mathbf{CL9} \vdash \lfloor H \rfloor$. It remains to show that $\lfloor H \rfloor = \lfloor E \rfloor$. Note that when floorifying E, $\check{P}^a_{\sqcup}$ gets replaced by P. But so does $\check{P}^a_b$ when floorifying H because, by our choice of b, $\check{P}^a_b$ is an isolated small molecule of H. Since the only difference between H and E is that H has $\check{P}^a_b$ where E has $\check{P}^a_{\sqcup}$, it is then obvious that indeed $\lfloor H \rfloor = \lfloor E \rfloor$.

Subcase 1.3: Neither of the above two conditions is satisfied. This means that in E all surface occurrences of large molecules are negative, and all surface occurrences of medium molecules are positive. Every such occurrence is an occurrence of a $\sqcup$-formula whose surface occurrences, as we remember, get replaced by $\bot$ when transferring from E to $\|E\|$; but the same happens to the corresponding occurrences of $\neg P$ or P in $\lfloor E \rfloor$ when transferring from $\lfloor E \rfloor$ to $\|\lfloor E \rfloor\|$. Based on this observation, with a little thought we can see that $\|\lfloor E \rfloor\|$ is "almost the same" as $\|E\|$; specifically, the only difference between these two formulas is that $\|\lfloor E \rfloor\|$ has $\bot$ where $\|E\|$ has isolated small molecules (positive or negative). Obviously this means that $\|\lfloor E \rfloor\|$ is a substitutional instance of $\|E\|$—the result of substituting, in the latter, each positive isolated small molecule by $\bot$ and the atomic part (the part under $\neg$) of each negative isolated small molecule by $\top$. As E is derived by Wait, $\|E\|$ is classically valid. Therefore $\|\lfloor E \rfloor\|$, as a substitutional instance of $\|E\|$, is also classically valid. So, we have:

$\lfloor E \rfloor$ *is stable.* (19)

Now consider an arbitrary formula H' that is the result of replacing in $\lfloor E \rfloor$ a surface occurrence of a subformula $G'_1 \sqcap \ldots \sqcap G'_n$ by G'_i for some $i \in \{1, \ldots, n\}$. Our goal is to show that

[17] Remember what was said earlier about the meaning of "negative occurrence" for small molecules.

CL9 $\vdash H'$ *(arbitrary H' satisfying the above condition).* (20)

The logical structure of E is the same as that of $\lfloor E\rfloor$, with the only difference that, wherever $\lfloor E\rfloor$ has general literals, E has molecules. Hence E has an occurrence of a subformula $G_1 \sqcap \ldots \sqcap G_n$ where $\lfloor E\rfloor$ has the above occurrence of $G'_1 \sqcap \ldots \sqcap G'_n$. Let then H be the result of replacing $G_1 \sqcap \ldots \sqcap G_n$ by G_i in E. Of course $H \in \vec{H}$. So, in view of (18), it would suffice to show (in order to verify (20)) that H is good and $H' = \lfloor H\rfloor$. Let us first see that H is good. When transferring from E to H, Condition (i) is inherited by H for the same or a similar reasons as in all of the previous cases. So is Condition (ii) because we are not creating any new non-surface occurrences. Furthermore, notice that $G_1 \sqcap \ldots \sqcap G_n$ is not a molecule, for otherwise in $\lfloor E\rfloor$ we would have a general literal rather than $G'_1 \sqcap \ldots \sqcap G'_n$. Hence, in view of Condition (ii), G_i is not a small or medium molecule. This means that, when transferring from E to H, we are not creating new independent/surface occurrences of any small or medium molecules, so that Conditions (iii) and (iv) are also inherited by H from E. To summarize, H is indeed good. Finally, it is also rather obvious that $H' = \lfloor H\rfloor$. The only case when we might have $H' \neq \lfloor H\rfloor$ would be if there was a small molecule $\check{P}^a_b$ isolated in E but not in H, or vice versa (so that the independent occurrence of that molecule in E would become P in $\lfloor E\rfloor$ and hence in H' but stay $\check{P}^a_b$ in $\lfloor H\rfloor$, or vice versa). But, as we observed just a little while ago, E and H do not differ in what independent/surface occurrences of what small molecules they have.

Next, consider an arbitrary formula H'' that is the result of replacing in $\lfloor E\rfloor$ a surface occurrence of a subformula $G''_0 \sqcap \ldots \sqcap G''_n$ by $G'_1 \sqcap \ldots \sqcap G''_n$. Our goal is to show that

CL9 $\vdash H''$ *(arbitrary H'' satisfying the above condition).* (21)

This case is very similar to the case handled in the previous paragraph. The logical structure of E is the same as that of $\lfloor E\rfloor$, so E has an occurrence of a subformula $G_0 \sqcap \ldots \sqcap G_n$ where $\lfloor E\rfloor$ has the above occurrence of $G''_0 \sqcap \ldots \sqcap G''_n$. Let then H be the result of replacing $G_0 \sqcap \ldots \sqcap G_n$ by $G_1 \sqcap \ldots \sqcap G_n$ in E. Of course $H \in \vec{H}$. Continuing arguing as in the previous paragraph, we find that H is good and that $H'' = \lfloor H\rfloor$, which, by (18), implies the desired (21).

Based on (19), (20) and (21), we find that $\lfloor E\rfloor$ is derivable in **CL9** by Wait.

The remaining two Cases 2 and 3 are about when E is derived by Choose or Switch from a premise H. Such an H turns out to be good and hence (by the induction hypothesis) its floorification **CL9**-provable. And, "almost always" $\lfloor E\rfloor$ follows from $\lfloor H\rfloor$ by Choose or Switch for the same reasons as E follows from H. An exception is the special case of Choose when H is the result of replacing in E a positive occurrence of a medium molecule $\check{P}^a_\sqcup$ by one of its disjuncts $\check{P}^a_b$ such that E has a negative independent occurrence of $\check{P}^a_b$. Using our earlier terms, this is a step signifying $\top$'s (final) decision to "match" the two P-based molecules. In this case, while $\lfloor E\rfloor$ does not follow from $\lfloor H\rfloor$ by Choose, it does so by Match. The secret is that the two P-based molecules are non-isolated small molecules in H and hence remain elementary literals in $\lfloor H\rfloor$, while they turn into general literals in $\lfloor E\rfloor$.

Case 2: E is derived by Choose. That is, we have **CL10** $\vdash H$, where H is the result of replacing in E a surface occurrence of a subformula $G = G_1 \sqcup \ldots \sqcup G_n$ by G_i for some $i \in \{1,\ldots,n\}$. Fix these formulas and this number i. Just as in Case 1 (statement (18)), based on the induction hypothesis, we have:

If H is good, then **CL9** $\vdash \lfloor H\rfloor$. (22)

We need to consider the following three subcases that cover all possibilities:

Subcase 2.1: G is not a molecule. Reasoning (almost) exactly as we did at the end of our discussion of Subcase 1.3, we find that H is good. Therefore, by (22), **CL9** $\vdash \lfloor H\rfloor$. Now, a little thought can convince us that $\lfloor E\rfloor$ follows from $\lfloor H\rfloor$ by Choose, so that **CL9** $\vdash \lfloor E\rfloor$.

Subcase 2.2: G is a negative large molecule $\neg\check{P}^1_\sqcup \sqcup \ldots \sqcup \neg\check{P}^m_\sqcup$. So, $G_i = \neg\check{P}^i_\sqcup$. A (now already routine for us) examination of Conditions (i)-(iv) reveals that each of these four conditions are inherited by H from E, so that H is good. Therefore, by (22), **CL9** $\vdash \lfloor H\rfloor$. Now, $\lfloor H\rfloor$ can be easily seen to be the same as $\lfloor E\rfloor$, and thus **CL9** $\vdash \lfloor E\rfloor$.

Subcase 2.3: G is a positive medium molecule $\check{P}^a_1 \sqcup \ldots \sqcup \check{P}^a_m$. So, $G_i = \check{P}^a_i$. There are two subsubcases to consider:

Subsubcase 2.3.1: E contains no independent occurrence of $\check{P}^a_i$. One can easily verify that H is good and that $\lfloor H\rfloor = \lfloor E\rfloor$. By (22), we then get the desired **CL9** $\vdash \lfloor E\rfloor$.

Subsubcase 2.3.2: E has an independent occurrence of $\check{P}^a_i$. Since E also has a positive independent occurrence of $\check{P}^a_\sqcup$, Condition (iv) implies that the above occurrence of $\check{P}^a_i$ in E is negative. This, in conjunction with Condition (iii), means that E does not have any other independent occurrences of $\check{P}^a_i$, and thus H has exactly two—one negative and one positive—independent occurrences of $\check{P}^a_i$. This guarantees that Condition (iii) is satisfied for H, because H and E only differ in that H has $\check{P}^a_i$ where E has $\check{P}^a_\sqcup$. Conditions (i) and (ii) are straightforwardly inherited by H from E. Finally, Condition (iv) also transfers from E to H because, even though H—unlike E—has a positive independent occurrence of $\check{P}^a_i$, it no longer has a positive independent occurrence of $\check{P}^a_\sqcup$ (which, by the same Condition (iv) for E, was unique in E). Thus, H is good and, by (22), **CL9** $\vdash \lfloor H\rfloor$. Note that since H is good, by Condition (ii), both of the independent occurrences of $\check{P}^a_i$ in it are surface occurrences. The same, of course, is true for the corresponding occurrences of $\check{P}^a_i$ and $\check{P}^a_\sqcup$ in E. Let us now compare $\lfloor E\rfloor$ with $\lfloor H\rfloor$. According to our earlier observation, $\check{P}^a_i$ only has one independent occurrence in E, i.e., $\check{P}^a_i$ is E-isolated. Hence the independent occurrence

of $\check{P}^a_i$, just as that of $\check{P}^a_\sqcup$, gets replaced by P when floorifying E. On the other hand, $\check{P}^a_i$ is no longer isolated in H, so the two independent occurrences of it stay as they are when floorifying H. Based on this observation, we can easily see that the only difference between $\lfloor E\rfloor$ and $\lfloor H\rfloor$ is that $\lfloor E\rfloor$ has the general atom P where $\lfloor H\rfloor$ has the (two occurrences of) elementary atom $\check{P}^a_i$. Since $\lfloor E\rfloor$ does not contain $\check{P}^a_i$ (because the only independent occurrence of it in E, as well as all large and medium P-based molecules, got replaced by P when floorifying E), and since we are talking about two—one positive and one negative—surface occurrences of P in $\lfloor E\rfloor$, we find that $\lfloor E\rfloor$ follows from $\lfloor H\rfloor$ by Match. We already know that $\mathbf{CL9}\vdash\lfloor H\rfloor$. Hence $\mathbf{CL9}\vdash\lfloor E\rfloor$.

Case 3: E is derived by Switch. That is, we have $\mathbf{CL10}\vdash H$, where H is the result of replacing in E a surface occurrence of a subformula $G_0\triangledown\ldots\triangledown G_k$ by $G_1\triangledown\ldots\triangledown G_k$. Just as in Cases 1 and 2, based on the induction hypothesis, we have:

If H is good, then $\mathbf{CL9}\vdash\lfloor H\rfloor$. (23)

Reasoning as in the previous cases, we further find that H is good, and thus, by (23), $\mathbf{CL9}\vdash\lfloor H\rfloor$. Now, a moment's thought convinces us that $\lfloor E\rfloor$ follows from $\lfloor H\rfloor$ by Switch, so that $\mathbf{CL9}\vdash\lfloor E\rfloor$.

Claim 1 is proven.

Now we are very close to finishing our proof of Lemma 9.1. Assume $\mathbf{CL9}\not\vdash F$. Let $\lceil F\rceil$ be the result of replacing in F all occurrences of each general atom $P\in\mathcal{P}$ by $\check{P}^{\sqcap}_{\sqcup}$. Obviously $\lceil F\rceil$ is good. Clearly we also have $\lfloor\lceil F\rceil\rfloor=F$, so that $\mathbf{CL9}\not\vdash\lfloor\lceil F\rceil\rfloor$. Therefore, by Claim 1, $\mathbf{CL10}\not\vdash\lceil F\rceil$. Hence, by Lemma 8.6, there is an interpretation † that interprets every elementary atom as a finitary predicate of arithmetical complexity Δ_2, such that

$\lceil F\rceil^\dagger$ *is not computable.* (24)

Let * be an interpretation such that:

- * agrees with † on all elementary atoms;
- * interprets each atom $P\in\mathcal{P}$ as $(\check{P}^{\sqcup}_{\sqcup})^\dagger$.

Clearly * interprets atoms as promised in our Lemma 9.1. It is also obvious that $F^*=\lceil F\rceil^\dagger$. Therefore, by (24), F^* is not computable, and the lemma is proven. □

10. A first-order extension of CL9

Here we introduce a first-order extension of **CL9** called **CL11**. The latter is also a conservative extension of logic **CL4** (proven to be sound and complete in [12]), obtained by augmenting its language with $\triangle$,$\triangledown$.

For each arity n, the language of **CL11** has infinitely many non-logical n-ary *elementary letters* $p, q, \ldots$and *general letters* $P, Q, \ldots$. An *elementary atom* is $p(t_1,\ldots,t_n)$, where p is an n-ary elementary letter and each t_i is a *term* (a variable or a constant). *General atoms* are defined similarly. As in the case of **CL9**, each interpretation * is required to interpret elementary atoms as elementary games, and general atoms as any static games. Such an interpretation * then extends to all formulas by commuting with the operation of substitution of variables by terms, and seeing all logical operators as the corresponding operations on games. There are some straightforward additional "admissibility" conditions on interpretations to avoid collisions of variables and some other unpleasant effects. We refer for details to [16] or [12].

The logical vocabulary of **CL11** is that of **CL9** plus the four quantifiers $\bigsqcap,\bigsqcup,\forall,\exists$. Formulas, that we refer to as **CL11-formulas**, are built from atoms, variables, constants and logical operators in the standard way. As before, negation is officially allowed to be applied only to non-logical atoms. For safety, we also require that no variable should have both free and bound occurrences in the same formula. The concepts of validity and uniform validity straightforwardly extend from **CL9**-formulas to **CL11**-formulas. So do most of the technical concepts defined earlier for the language of **CL9**. Two of those still need to be slightly redefined. Namely, a **surface occurrence** now is an occurrence that is not in the scope of a choice connective or a choice quantifier and is not in the tail of any sequential subformula. And the **elementarization** of a formula F now means the result of replacing in the capitalization of F every surface occurrence of the form $G_1\sqcap\ldots\sqcap G_n$ or $\bigsqcap xG$ by $\top$, every surface occurrence of the form $G_1\sqcup\ldots\sqcup G_n$ or $\bigsqcup xG$ by $\bot$, and every positive surface occurrence of each general literal by $\bot$. Finally, a formula is said to be **stable** iff its elementarization is a valid formula of classical first-order logic; otherwise it is **instable**.

In the above language, the logic is axiomatized as follows:

Definition 10.1. The rules of inference of **CL11** are:

Wait: $\vec{H}\vdash F$, where F is stable and $\vec{H}$ is a set of formulas satisfying the following three conditions:

- whenever F has a surface occurrence of a subformula $G_1\sqcap\ldots\sqcap G_n$, for each $i\in\{1,\ldots,n\}$, $\vec{H}$ contains the result of replacing that occurrence in F by G_i;
- whenever F has a surface occurrence of a subformula $G_0\triangle G_1\triangle\ldots\triangle G_n$, $\vec{H}$ contains the result of replacing that occurrence in F by $G_1\triangle\ldots\triangle G_n$;

- whenever F has a surface occurrence of a subformula $\sqcap xG(x)$, $\vec{H}$ contains the result of replacing that occurrence in F by $G(y)$, where y is a variable not occurring in F.

$\sqcup$-Choose: $H \vdash F$, where H is the result of replacing in F a surface occurrence of a subformula $G_1 \sqcup \ldots \sqcup G_n$ by G_i for some $i \in \{1,\ldots,n\}$.

$\bigsqcup$-Choose: $H \vdash F$, where H is the result of replacing in F a surface occurrence of a subformula $\bigsqcup xG(x)$ by $G(t)$, where t is a term with no bound occurrence in F.

Switch: $H \vdash F$, where H is the result of replacing in F a surface occurrence of a subformula $G_0 \triangledown G_1 \triangledown \ldots \triangledown G_n$ by $G_1 \triangledown \ldots \triangledown G_n$.

Match: $H \vdash F$, where H is the result of replacing in F two—one positive and one negative — surface occurrences of some n-ary general letter P by an n-ary non-logical elementary letter p that does not occur in F.

There is every reason to expect that the already known soundness and completeness theorems for **CL9** and **CL4** extend to their common extension **CL11**. A proof of this fact can be obtained by combining the techniques and ideas employed in the above two soundness/completeness proofs. The author does not see any reasons why such an (almost mechanical) combination would not work. This job has to be actually done though, and until then the following statement should be officially considered only a conjecture rather than a theorem.

Conjecture 10.2. **CL11** $\vdash F$ *iff* F *is valid (any* **CL11***-formula* F*). Furthermore:*

(a) There is an effective procedure that takes a **CL11***-proof of an arbitrary formula* F *and constructs an HPM* $\mathcal{H}$ *such that, for every interpretation* **,* $\mathcal{H}$ *computes* F^**.*

(b) If **CL11** $\not\vdash F$*, then* F^* *is not computable for some interpretation* * *that interprets all elementary atoms of* F *as finitary predicates of arithmetical complexity* Δ_2*, and interprets all general atoms of* F *as problems of the form* $(A^1_1 \sqcup \ldots \sqcup A^1_m) \sqcap \ldots \sqcap (A^m_1 \sqcup \ldots \sqcup A^m_m)$*, where each* A^j_i *is a finitary predicate of arithmetical complexity* Δ_2*.*

Theorem 10.3. *The* $\forall,\exists$*-free fragment of* **CL11** *(i.e., the set of all* $\forall,\exists$*-free theorems of* **CL11***) is decidable in polynomial space.*

Proof. This is similar to the corresponding theorem for **CL4** proven in [12]—the presence of $\triangle,\triangledown$ in formulas hardly creates any differences. So, it would be sufficient to give just a schematic outline of the proof idea for our present theorem. A polynomial-space decision algorithm for **CL11**-provability of a $\forall,\exists$-free formula F is a recursive one. At each level of recursion, it tests all possible premises (for any of the five possible rules) for F, calling itself on those premises. In each case, there is only a finite number of premises to test. Strictly speaking, there are infinitely many possible premises for Match. However, those premises only differ from each other in selecting a fresh elementary letter to replace two occurrences of a general letter. Of course, one choice of such a letter can yield a provable premise iff any other choice can, so considering only one premise would be sufficient. Similarly for the rules associated with $\sqcap$ and $\bigsqcup$ ($\sqcap$-Choose and Wait). Each time the algorithm deals with Wait, it has to test whether the conclusion is stable. While stability of **CL11**-formulas is generally undecidable, for $\forall,\exists$-free **CL11**-formulas it can be easily seen to be decidable in linear space. Each level of recursion thus only takes polynomial space. And the depth of recursion is limited by the size of the formula. So, the whole algorithm runs in polynomial space. □

We close this section with two examples that can help us get some syntactic feel of **CL11** and appreciate its value as a problem-solving tool.

Example 10.4. Remember formula (5), claimed to be valid in Section 2.7. Below is a **CL11**-proof of it:

1. $\neg q(y) \vee P(y) \vee q(y)$ (from {} by Wait)
2. $\neg q(y) \vee \Big(\neg p(y) \triangle P(y)\Big) \vee q(y)$ (from {1} by Wait)
3. $\neg P(y) \vee \Big(\neg p(y) \triangle P(y)\Big) \vee P(y)$ (from 2 by Match)
4. $\neg P(y) \vee \Big(\neg p(y) \triangle P(y)\Big) \vee \Big(P(y) \sqcup \neg P(y)\Big)$ (from 3 by $\sqcup$-choose)
5. $\neg P(y) \vee q(y) \vee \neg q(y)$ (from {} by Wait)
6. $\Big(p(y) \triangle \neg P(y)\Big) \vee q(y) \vee \neg q(y)$ (from {5} by Wait)
7. $\Big(p(y) \triangle \neg P(y)\Big) \vee P(y) \vee \neg P(y)$ (from 6 by Match)
8. $\Big(p(y) \triangle \neg P(y)\Big) \vee P(y) \vee \Big(P(y) \sqcup \neg P(y)\Big)$ (from 7 by $\sqcup$-Choose)
9. $\Big(p(y) \triangle \neg P(y)\Big) \vee \Big(\neg p(y) \triangle P(y)\Big) \vee \Big(P(y) \sqcup \neg P(y)\Big)$ (from {4,8} by Wait)

10. $\Big(P(y)\triangle\neg P(y)\Big) \vee \Big(\neg P(y)\triangle P(y)\Big) \vee \Big(P(y) \sqcup \neg P(y)\Big)$ (from 9 by Match)

11. $\Big(P(y)\triangle\neg P(y)\Big) \vee \bigsqcup x\Big(\neg P(x)\triangle P(x)\Big) \vee \Big(P(y) \sqcup \neg P(y)\Big)$ (from 10 by $\bigsqcup$-Choose)

12. $\bigsqcup x\Big(P(x)\triangle\neg P(x)\Big) \vee \bigsqcup x\Big(\neg P(x)\triangle P(x)\Big) \vee \Big(P(y) \sqcup \neg P(y)\Big)$ (from 11 by $\bigsqcup$-Choose)

13. $\bigsqcup x\Big(P(x)\triangle\neg P(x)\Big) \vee \bigsqcup x\Big(\neg P(x)\triangle P(x)\Big) \vee \bigsqcap x\Big(P(x) \sqcup \neg P(x)\Big)$ (from {12} by Wait)

Example 10.5. Let us now see the **CL11**-provability of formula (6) from Section 2.7. Below $E \not\leftrightarrow F$ is an abbreviation of $\neg(E \leftrightarrow F)$, i.e., of $(E \wedge \neg F) \vee (F \wedge \neg E)$:

1. $\Big(q(z) \not\leftrightarrow p(v)\Big) \vee \neg p(v) \vee q(z)$ (from {} by Wait)

2. $\Big(q(z) \not\leftrightarrow p(v)\Big) \vee \neg p(v) \vee \Big(\neg q(z)\triangledown q(z)\Big)$ (from 1 by Switch)

3. $\Big(q(z) \not\leftrightarrow p(v)\Big) \vee \Big(p(v)\triangle\neg p(v)\Big) \vee \Big(\neg q(z)\triangledown q(z)\Big)$ (from {2} by Wait)

4. $\Big(q(z) \not\leftrightarrow p(v)\Big) \vee \bigsqcup x\Big(p(x)\triangle\neg p(x)\Big) \vee \Big(\neg q(z)\triangledown q(z)\Big)$ (from 3 by $\bigsqcup$-Choose)

5. $\bigsqcap y\Big(q(z) \not\leftrightarrow p(y)\Big) \vee \bigsqcup x\Big(p(x)\triangle\neg p(x)\Big) \vee \Big(\neg q(z)\triangledown q(z)\Big)$ (from {4} by Wait)

6. $\bigsqcup x\bigsqcap y\Big(q(x) \not\leftrightarrow p(y)\Big) \vee \bigsqcup x\Big(p(x)\triangle\neg p(x)\Big) \vee \Big(\neg q(z)\triangledown q(z)\Big)$ (from 5 by $\bigsqcup$-Choose)

7. $\bigsqcup x\bigsqcap y\Big(q(x) \not\leftrightarrow p(y)\Big) \vee \bigsqcup x\Big(p(x)\triangle\neg p(x)\Big) \vee \bigsqcap x\Big(\neg q(x)\triangledown q(x)\Big)$ (from {6} by Wait)

Appendix

A. On abstract resource semantics

Abstract resource semantics, introduced in [9], is a companion of computability logic. One could probably characterize it as a "lazy" and naive form of the semantics of CL. Here we outline it in very informal terms.

All rules of our systems—**CL9**, **CL9**°, **CL11**—pretty much look like (the effects of) moves in associated games. The only exception is Match, which has no direct counterpart in games. The central idea of abstract resource semantics is to make Match a legitimate move (by $\top$) in its own right, called *allocation*. [9] provided plenty of intuitive explanations, showing how this sort of an approach yields a direct materialization of the resource intuitions traditionally (and somewhat wrongly) associated with linear logic, in the "can you get both a candy and an ice cream for one dollar?" style.

The language that abstract resource semantics deals with is the same as the language of computability logic (possibly with just minor differences such as considering hyperformulas instead of formulas), with two sorts—elementary and general—of atoms. And, as in computability logic, formulas are understood as games. There are three main differences.

One, rather minor, difference is that abstract resource semantics prefers to see each position not as a sequence of moves but rather the formula—more precisely, the hyperformula—representing the game to which such a sequence brings the original game down. This is an approach taken in all pre-CL papers by the present author, most notably in [4]. For example, the position $\langle\top 1.2\rangle$ of game $\Big(P \sqcup (\underline{Q}\triangle R)\Big) \wedge S$ will be simply seen as the hyperformula $(\underline{Q}\triangle R) \wedge S$, and the move 1.2 by $\top$ as the action of turning $\Big(P \sqcup (\underline{Q}\triangle R)\Big) \wedge S$ into $(\underline{Q}\triangle R) \wedge S$. The further move 1.§ by $\bot$ will be seen as turning $(\underline{Q}\triangle R) \wedge S$ into $(Q\triangle\underline{R}) \wedge S$, the further move 1.§ by $\top$ will be seen as turning $(Q\triangle\underline{R}) \wedge S$ into $\underline{R} \wedge S$, etc.[18]

The second difference is that, while computability logic treats formulas as schemata of games—that is, syntactic expressions that become games only after an interpretation * is applied to them—abstract resource semantics simply sees formulas as full-fledged games ("*abstract resources*"). It does not apply or appeal to interpretations, essentially meaning that it has a concept of validity but no concept of truth.

The third, most important difference, as already noted, is that, along with all "ordinary" moves permitted in computability logic, abstract resource semantics allows an additional sort of a move called *(resource) allocation*. It consists of pairing one positive and one negative occurrence of a general atom P. The effect of such a move can be stipulated to be the result of replacing, in the original formula, the two occurrences of P by the hybrid atom P_q for some fresh elementary atom q. That is, allocation is a direct counterpart of (or the same as) Match—or, more precisely, Match°. If no recurrence operators and

[18] Note the minor difference from how we treated catch-up switch moves in Section 6: such moves had no effect there, and $(Q\triangle\underline{R}) \wedge S$ would remain $(Q\triangle\underline{R}) \wedge S$ instead of turning into $\underline{R} \wedge S$.

no parallel and sequential quantifiers are present, every game will only last a finite number of steps, and will be considered won by $\top$ iff the final formula/position is stable. It is almost immediately obvious (only for an expert, of course) that our systems **CL9** and **CL11** continue to be sound and complete with respect this semantics. In fact, proving such soundness and completeness would be by an order of magnitude easier than proving soundness and completeness with respect to the semantics of CL, because the relevant parts of abstract resource semantics are essentially directly "read" from the rules of those systems.

The above approach easily extends to the fragment of the language of CL containing parallel and sequential recurrences, as well as all sorts of quantifiers. The only difference will be that now the "final" ("limit") formula, whose stability determines the outcome of the game, may be infinite, containing infinite parallel conjunctions and/or disjunctions. Furthermore, a sequential subformula of the "final" formula may have no particular underlined component due to an infinite number of leading switches made in it. Such a subformula should be replaced by $\bot$ (if it was a $\triangledown$, $\bigtriangledown$ or $\curlyvee$-subformula) or $\top$ (if it was a $\triangle$, $\bigtriangleup$ or $\curlywedge$-subformula). Then extending the concept of stability to such limit formulas presents no problem.

A relatively non-trivial step is further extending the approach to branching recurrences as well. Those familiar with the relevant pieces of literature (such as Section 4.6 of [16]) would remember that the effect of making a "replicative move" in $\overset{\downarrow}{\circ}E$ or $\overset{\uparrow}{\circ}E$ is turning it into $\overset{\downarrow}{\circ}(E \circ E)$ or $\overset{\uparrow}{\circ}(E \circ E)$, respectively. Adding $\circ$ to the language that we consider is not necessary though, as $\overset{\downarrow}{\circ}(A \circ B)$ is equivalent to $\overset{\downarrow}{\circ}A \wedge \overset{\downarrow}{\circ}B$ and $\overset{\uparrow}{\circ}(A \circ B)$ is equivalent to $\overset{\uparrow}{\circ}A \vee \overset{\uparrow}{\circ}B$. So, we can stipulate that a replicative move turns $\overset{\downarrow}{\circ}E$ into $\overset{\downarrow}{\circ}E \wedge \overset{\downarrow}{\circ}E$ and $\overset{\uparrow}{\circ}E$ into $\overset{\uparrow}{\circ}E \vee \overset{\uparrow}{\circ}E$. And the effect of a non-replicative move α within $\overset{\downarrow}{\circ}E$ or $\overset{\uparrow}{\circ}E$ is simply replacing E by the effect of α on E. In this respect, branching operators do not differ from any other operators. What makes the case of branching operators special is that general atoms that are in the scope of such operators will have to be reallocated over and over again (as the associated replicative moves create two copies of the argument of $\overset{\downarrow}{\circ},\overset{\uparrow}{\circ}$). Reallocations here should follow the same constraint as all allocations do—specifically, (re)allocation can only take place between a positive and a negative occurrence of the *same* atom, whether such an atom is an original general atom P or a previously already allocated atom which now looks like $P_{\vec{q}}$. When two occurrences of such a $P_{\vec{q}}$ are allocated to each other, they become $P_{\vec{q},r}$ for some fresh r. Thus, here we may get an infinite "final" formula not only because of infinitely many subformulas, but also because of hybrid atoms $P_{\vec{s}}$ with infinite subscripts $\vec{s}$. Two such atoms $P_{\vec{p}}$ and $Q_{\vec{q}}$ counting as the same iff $P = Q$ and $\vec{p} = \vec{q}$, the concept of stability then painlessly extends to the present case as well.

The approach called the *logic of tasks*, introduced and elaborated in [4], was essentially nothing but the above-outlined abstract resource semantics limited to the language of (not yet officially born) CL without general atoms and without branching and sequential operators. In the absence of general atoms, the logic of tasks did not employ allocation. On the other hand, [9] dealt with a language with general atoms and used allocation as a basic semantical concept. But the logical vocabulary for which abstract resource semantics was fully defined (and, most importantly, well-motivated intuitively) was limited to $\{\neg, \wedge, \vee\}$. The extension of abstract resource semantics to the full language of CL outlined in the present appendix is a mechanical combination of the approaches of [9] and [4], extended further to the sequential and branching operators that were present in neither [9] nor [4].

Both [4] and [9] outlined potential applications of abstract resource semantics in resource-based planning systems. It has been also argued that planning systems based on such a semantics are immune to the notorious frame problem and the knowledge preconditions problem. That outline still needs a further materialization and elaboration though. Until that is done, abstract resource semantics (unlike the semantics of CL) should probably be treated as a technical tool rather than a semantics in its own right. As such, it may be very useful in proving various theorems about computability logic.

B. Proof of Theorem 4.4

Lemma B.1

1. Assume $A_0, \ldots, A_n$ are constant static games, Σ is a $\wp$-delay of Γ, and Σ is a $\wp$-illegal run of $A_0 \triangle \ldots \triangle A_n$. Then Γ is also a $\wp$-illegal run of $A_0 \triangle \ldots \triangle A_n$.

2. Similarly for $A_0 \triangle A_1 \triangle A_2 \triangle \ldots$.

Proof. We will prove this lemma by induction on the length of the shortest illegal initial segment of Σ.

Clause 1. Assume the conditions of clause 1 of the lemma. We want to show that Γ is a $\wp$-illegal run of $A_0 \triangle \ldots \triangle A_n$. Let $\langle \Psi, \wp\alpha \rangle$ be the shortest ($\wp$-) illegal initial segment of Σ. Let $\langle \Phi, \wp\alpha \rangle$ be the shortest initial segment of Γ containing all the $\wp$-labeled moves[19] of $\langle \Psi, \wp\alpha \rangle$. If Φ is a $\wp$-illegal position of $A_0 \triangle \ldots \triangle A_n$, then so is Γ and we are done. Therefore, for the rest of the proof, we assume that

$$\Phi \textit{ is not a } \wp\textit{-illegal position of } A_0 \triangle \ldots \triangle A_n. \tag{B.1}$$

Let Θ be the sequence of those $\neg\wp$-labeled moves of Ψ that are not in Φ. Obviously

$$\langle \Psi, \wp\alpha \rangle \textit{ is a } \wp\textit{-delay of } \langle \Phi, \wp\alpha, \Theta \rangle. \tag{B.2}$$

[19] In this context, different occurrences of the same labmove count as different labmoves. So, a more accurate phrasing would be "as many $\wp$-labeled moves as..." instead "all the $\wp$-labeled moves of ...".

We also claim that

Φ *is a legal position of* $A_0\vartriangle\ldots\vartriangle A_n$. (B.3)

Indeed, suppose this was not the case. Then, by (B.1), Φ should be $\neg\wp$-illegal. This would make Γ a $\neg\wp$-illegal run of $A_0\vartriangle\ldots\vartriangle A_n$ with Φ as an illegal initial segment which is shorter than $\langle\Psi,\wp\alpha\rangle$. Then, by the induction hypothesis, any run for which Γ is a $\neg\wp$-delay, would be $\neg\wp$-illegal. But, as observed in Lemma 4.6 of [5], the fact that Σ is a $\wp$-delay of Γ implies that Γ is a $\neg\wp$-delay of Σ. So, Σ would be $\neg\wp$-illegal, which is a contradiction because, according to our assumption, Σ is $\wp$-illegal.

We are continuing our proof. There are three possible reasons to why $\langle\Psi,\wp\alpha\rangle$ is an illegal (while Ψ being legal) position of $A_0\vartriangle\ldots\vartriangle A_n$:

Reason 1: α does not have the form § or $.\beta$. Then, in view of (B.3), $\langle\Phi,\wp\alpha\rangle$ is a $\wp$-illegal position of $A_0\vartriangle\ldots\vartriangle A_n$. As $\langle\Phi,\wp\alpha\rangle$ happens to be an initial segment of Γ, the latter then is a $\wp$-illegal run of $A_0\vartriangle\ldots\vartriangle A_n$.

Reason 2: $\alpha=$ §, and either $\wp=\bot$ and the $\bot$-degree of Ψ is n, or $\wp=\top$ and the $\top$-degree of Ψ equals the $\bot$-degree of Ψ. In either case, with (B.3) in mind, $\langle\Phi,\wp\alpha\rangle$ can be seen to be a $\wp$-illegal position of $A_0\vartriangle\ldots\vartriangle A_n$. Hence, as $\langle\Phi,\wp\alpha\rangle$ is an initial segment of Γ, the latter is a $\wp$-illegal run of $A_0\vartriangle\ldots\vartriangle A_n$.

Reason 3: $\alpha=.\beta$, the $\wp$-degree of $\wp\alpha$ is $i\in\{0,\ldots,n\}$, and $\langle\Psi^{\#i},\wp\beta\rangle\notin\mathbf{Lr}^{A_i}$. That is, $\langle\Psi,\wp\alpha\rangle^{\#i}$ is a $\wp$-illegal position of A_i. (B.2) obviously implies that $\langle\Psi,\wp\alpha\rangle^{\#i}$ is a $\wp$-delay of $\langle\Phi,\wp\alpha,\Theta\rangle^{\#i}$. Therefore, since A_i is static, clause 1 of Lemma 6.6 yields that $\langle\Phi,\wp\alpha,\Theta\rangle^{\#i}$ is a $\wp$-illegal position of A_i. Notice that $\langle\Phi,\wp\alpha,\Theta\rangle^{\#i}=\langle\Phi^{\#i},\wp\beta,\Theta^{\#i}\rangle$. A $\wp$-illegal position will remain $\wp$-illegal after removing a block of $\neg\wp$-labeled moves (in particular, $\Theta^{\#i}$) at the end of it. Hence, $\langle\Phi^{\#i},\wp\beta\rangle$ is a $\wp$-illegal position of A_i. In view of (B.3), this implies that $\langle\Phi,\wp\alpha=\wp.\beta\rangle\notin\mathbf{Lr}^{A_0\vartriangle\ldots\vartriangle A_n}$, so that $\langle\Phi,\wp\alpha\rangle$ is a $\wp$-illegal position of $A_0\vartriangle\ldots\vartriangle A_n$, and then so is Γ because $\langle\Phi,\wp\alpha\rangle$ is an initial segment of it.

Clause 2. The reasoning here is virtually the same as in the proof of clause 1. The only difference (that makes the present case simpler) is that, in "*Reason 2*", the condition "$\wp=\bot$ and the $\bot$-degree of Ψ is n" does not need to be considered. □

Since $\vartriangle x,\triangledown x,\barwedge,\barvee$ are nothing but sequential conjunctions/disjunctions, there is no need to separately consider them when proving Theorem 4.4. Furthermore, since $\triangledown$ is expressible in terms of $\vartriangle$ and $\neg$ (with $\neg$ already known to preserve the static property), considering only $\vartriangle$ would be sufficient. For simplicity, here we restrict ourselves to the n-ary case of $\vartriangle$. Adapting our argument to the infinite case of $\vartriangle$ does not present a problem.

Assume that $A_0,\ldots,A_n$ are static constant games, $\mathbf{Wn}^{A_0\vartriangle\ldots\vartriangle A_n}\langle\Gamma\rangle=\wp$ and Σ is a $\wp$-delay of Γ. Our goal is to show that $\mathbf{Wn}^{A_0\vartriangle\ldots\vartriangle A_n}\langle\Sigma\rangle=\wp$.

If Σ is a $\neg\wp$-illegal run of $A_0\vartriangle\ldots\vartriangle A_n$, then it is won by $\wp$ and we are done. So, assume that Σ is not $\neg\wp$-illegal. Lemma 4.6 of [5] asserts that, if Σ is a $\wp$-delay of Γ, then Γ is a $\neg\wp$-delay of Σ. So, by Lemma B.1, our Γ cannot be $\neg\wp$-illegal, for otherwise so would be Σ. Γ also cannot be $\wp$-illegal, because otherwise it would not be won by $\wp$. Consequently, Σ cannot be $\wp$-illegal either, for otherwise, by Lemma B.1, Γ would be $\wp$-illegal. Thus, we have narrowed down our considerations to the case when both Γ and Σ are legal runs of $A_0\vartriangle\ldots\vartriangle A_n$.

$\mathbf{Wn}^{A_0\vartriangle\ldots\vartriangle A_n}\langle\Gamma\rangle=\wp$, together with $\Gamma\in\mathbf{Lr}^{A_0\vartriangle\ldots\vartriangle A_n}$, implies that, where $i\,(0\le i\le n)$ is the $\bot$-degree of Γ, $\Gamma^{\#i}$ is a $\wp$-won run of A_i. Taking into account that $\Sigma^{\#i}$ is obviously a $\wp$-delay of $\Gamma^{\#i}$ and that A_i is static, the above, in turn, implies that $\Sigma^{\#i}$ is a $\wp$-won run of A_i, which, taking into account that $\Sigma\in\mathbf{Lr}^{A_0\vartriangle\ldots\vartriangle A_n}$, means nothing but that Σ is a $\wp$-won run of $A_0\vartriangle\ldots\vartriangle A_n$.

References

[1] S. Abramsky, R. Jagadeesan, Games and full completeness for multiplicative linear logic, Journal of Symbolic Logic, 59 (2) (1994) 543–574.
[2] A. Blass, Degrees of indeterminacy of games, Fundamenta Mathematicae 77 (1972) 151–166.
[3] A. Blass, A game semantics for linear logic, Annals of Pure and Applied Logic 56 (1992) 183–220.
[4] G. 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 123 (2003) 1–99.
[6] G. Japaridze, Propositional computability logic I, ACM Transactions on Computational Logic 7 (2) (2006) 302–330.
[7] G. Japaridze, Propositional computability logic II, ACM Transactions on Computational Logic 7 (2) (2006) 331–362.
[8] G. Japaridze, From truth to computability I, Theoretical Computer Science 357 (2006) 100–135.
[9] G. Japaridze, Introduction to cirquent calculus and abstract resource semantics, Journal of Logic and Computation 16 (2006) 489–532.
[10] G. Japaridze, Computability logic: a formal theory of interaction, in: D. Goldin, S. Smolka, P. Wegner (Eds.), Interactive Computation: The New Paradigm, Springer-Verlag, Berlin, 2006, pp. 183–223.
[11] G. Japaridze, The logic of interactive Turing reduction, Journal of Symbolic Logic 72 (1) (2007) 243–276.
[12] G. Japaridze, From truth to computability II, Theoretical Computer Science 379 (2007) 20–52.
[13] G. Japaridze, Intuitionistic computability logic, Acta Cybernetica 18 (1) (2007) 77–113.
[14] G. Japaridze, The intuitionistic fragment of computability logic at the propositional level, Annals of Pure and Applied Logic 147 (3) (2007) 187–227.
[15] G. Japaridze, Cirquent calculus deepened, Journal of Logic and Computation 18 (6) (2008) 983–1028.
[16] G. Japaridze, In the beginning was game semantics, in: O. Majer, A.-V. Pietarinen, T. Tulenheimo (Eds.), Games: Unifying Logic, Language and Philosophy, Springer Verlag, Berlin, 2009, pp. 249–350.
[17] G. Japaridze, Many concepts and two logics of algorithmic reduction, Studia Logica, in press.
[18] A.N. Kolmogorov, Zur Deutung der intuitionistischen Logik, Mathematische Zeitschrift 35 (1932) 58–65.