Concurrent Dynamic Logic

DAVID PELEG

The Weizmann Institute of Science, Rehovot, Israel

Abstract. In this paper concurrent dynamic logic (CDL) is introduced as an extension of dynamic logic tailored toward handling concurrent programs. Properties of CDL are discussed, both on the propositional and first-order level, and the extension is shown to possess most of the desirable properties of DL. Its relationships with the μ-calculus, game logic, DL with recursive procedures, and PTIME are further explored, revealing natural connections between concurrency, recursion, and alternation.

Categories and Subject Descriptors: F.1.2 [**Computation by Abstract Devices**]: Modes of Computation—*alternation and nondeterminism*; *parallelism*; F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs—*logics of programs*; *dynamic logics*; F.3.3 [**Logics and Meanings of Programs**]: Studies of Program Constructs—*control primitives*; *program and recursion schemes*

General Terms: Languages, Theory, Verification

Additional Key Words and Phrases: Concurrency, DL

1. *Introduction*

Dynamic logic (DL) [5, 6] has been widely recognized during the past few years as an elegant and powerful tool for reasoning about programs. It has received considerable attention, and many of its aspects have been thoroughly investigated (cf. [8]). DL has proved particularly suitable for clarifying the notion of *nondeterminism*, and a large body of research concerns the effects of this property (or the lack thereof) in programs. On the other hand, the equally important notion of *concurrency* is not captured in DL in a satisfactory manner.

Consequently, there have been several attempts to develop systems based on DL that provide a more appropriate treatment of concurrency. Most of these adopt the view that, in order to discuss concurrency properly, it is necessary to allow the logic to express assertions regarding the *ongoing* behavior of a program, that is, to refer to the intermediate states of a computation path. This has led to a variety of process logics [11, 13, 24, 26, 41], that combine the features of DL and those of temporal logic [32]. Other attempts used the "shuffle" construct to realize concurrency [25].

There is, however, another approach to concurrency, which has proved successful in several areas of the theory of computation, but, which, although being quite natural, has not yet been considered in the context of dynamic logic. This approach views concurrency in its purest form as the *dual* notion of nondeterminism. For example, let us illustrate a process by a tree whose arcs represent "atomic actions." Nondeterminism is introduced by allowing a node to split into several branches,

Author's current address: Department of Computer Science, Stanford University, Stanford, CA 94305.

and letting the process choose between the different possible continuations. Analogously, concurrency is obtained by again splitting a node into several branches, but requiring the process to execute *all* possible continuations. This is basically the classical concept of and/or decomposition, which occurs widely in logic, game theory, etc.

This approach to computation was introduced initially by Chandra et al. in [2], through the mechanism of *alternating Turing machines* (ATMs); the computation tree of an ATM can be represented by an and/or tree. ATMs have proved to be a very useful tool in complexity theory, and have been followed by several attempts to develop programming tools incorporating this duality, like Harel's *and/or programs* [7], the language *Ind* of [9], and, in a way, the new trend of logic programming and *Prolog* [3, 18, 35].

This view of concurrency suggests a new viable approach toward incorporating concurrency into dynamic logic. We still concern ourselves with essentially only the input–output behavior of programs, but rather than consider computation *paths*, we achieve concurrency by allowing programs to execute from a state to a *set of states*. In this paper we propose a natural extension of DL based on this approach, named *concurrent dynamic logic* (CDL). (Its propositional and quantified, that is, first-order, versions are named CPDL and CQDL, respectively.) The logic captures the nondeterminism/concurrency duality by admitting a new *concurrency connective*, $\cap$, for programs. The program $\alpha \cap \beta$ is interpreted as "α and β executed in parallel." Therefore, the formula $\langle \alpha \cap \beta \rangle A$ reads "α and β can be executed in parallel so that upon termination (in either computation path) A holds," which is actually $\langle \alpha \rangle A \wedge \langle \beta \rangle A$. (This use of $\cap$ should not be confused with either the language-theoretic intersection operator, or the "relation-theoretic" one appearing in [8, sect. 2.5.5]; it is basically the $\wedge$ operator of Manna [20] and Chandra [1], although the overall semantics is different.) By this we supplement the static and/or duality in the formula component of DL with a dynamic concurrency/nondeterminism duality in its program component.

Recently, a new *game logic* was introduced by Parikh in [27] and [28], as a propositional-level logic for handling topics like games, concurrency, and alternation. Although the general features of game logic resemble those of CDL, and indeed our present work was motivated and insipired to a great extent by [27] and [28], there are several differences. Conceptually, game logic views the subject from a different angle, as implied by its name. Technically, the main difference is in the use of a strong *dual* operator on programs in game logic. This essentially provides a dual form for any program connective, and not just for $\cup$, and thus corresponds to a wider notion than pure concurrency. The additional strength of game logic is further reflected in the fact that it is not yet known to have a complete axiom system. Another technical difference is a certain monotonicity requirement imposed on programs in game logic. We discuss the matter in more detail in Section 2.2.

A slightly different approach for adding alternation to a logic of programs is proposed in [43], in the framework of extended temporal logic.

Our main findings can be grouped as follows. To begin with, CDL is shown to be strictly more expressive than DL on the propositional level. We conjecture that the same holds on the first-order level too, but we can show this only for the restricted version of quantified dynamic logic (QDL) with "quantifier-free" tests.

Next come several results that serve to clarify the relationships between concurrency, recursion, and alternation. The close relationships between versions of dynamic logic and the μ-calculus have caused us to seek the precise version of the

μ-calculus that corresponds to CDL. In general, we consider two restrictions on the μ-calculus. The first is *continuity*, that is, an expression $\mu X.F(X)$ allows X to appear under no negations at all, and the second is *simplicity*, that is, least fixpoints are not allowed to interleave (e.g., an expression $\mu X(\cdots \mu Y(\cdots X \cdots) \cdots)$ is forbidden).

On the propositional level, CPDL lies in expressiveness somewhere between the *simple continuous* and the *continuous* sublanguage of L_μ, defined later; on the first-order level, CQDL with random assignments, **ran**-CQDL, is equivalent to *continuous* μ-calculus. The second result is obtained by equating **ran**-CQDL to **ran**-QDL$_{\text{proc}}$, that is, QDL that allows recursive procedures with parameters. Without random assignment, QDL$_{\text{proc}}$ is strictly more expressive than CQDL.

Over finite ordered structures, **ran**-CQDL defines precisely PTIME, in the sense that a set of such structures is in PTIME iff it is the graph of some closed **ran**-CQDL formula. This last observation is in accordance with several other such facts appearing in [10]; in particular, that ordinary **ran**-QDL is equivalent to NLOGSPACE. One would indeed have expected the introduction of alternation to raise the complexity by one level. In fact, this justifies our view that CDL is to DL exactly what ATMs are to NTMs.

All comparisons in expressive power are restricted to *sequential models*, which are models in which all atomic programs are sequential, that is, lead from states to (single) states. By adopting such a restriction on models of CDL, we make them compatible with those of all aforementioned logics, such as regular DL and the μ-calculus. This is actually quite a natural requirement, since in most parallel programming languages atomic programs are sequential, and parallelism is obtained by higher level combinators only.

Finally, we consider all of the basic results obtained for standard dynamic logic that concern validity and completeness, and reestablish them for the concurrent case with the appropriate modifications. Specifically, on the propositional level, we provide the general version of CPDL with a complete axiom system and consequently deduce its decidability. (In contrast, Parikh [28] proves completeness for "dual-free" game logic, but the axiom system proposed therein for the full-game logic is not yet known to be complete.)

On the first-order level, validity is observed to remain Π_1^1-complete, and the issue of axiomatization is treated along the lines of [8] for QDL. That is, we provide results of completeness for termination assertions and infinitary completeness in the uninterpreted case, and relative arithmetical completeness in the interpreted case.

The overall weight of these results gives us reason to believe that CDL is a natural, reasonable, and interesting extension of DL, providing adequate treatment of the issue of concurrency, yet maintaining the desirable properties of dynamic logic.

Taking a more general view and trying to judge the appropriateness of CDL for discussion of currently available concurrent programming languages, it is immediately apparent that CDL suffers from the absence of any communication mechanisms; processes of CDL are totally independent and mutually ignorant, once split. In related work [30], we extend CDL further, so as to account also for this aspect.

The rest of the paper is divided into two main sections, discussing the propositional and first-order levels, respectively, according to the outlined points.

2. *The Propositional Level*

2.1 CONCURRENT PDL. The syntax of CPDL is just that of PDL, with the addition of a concurrency operator $\cap$ on programs. Thus, we have atomic formulas P_i, atomic programs a_i, and the construction rules:

(1) Every P_i is a formula.
(2) If A and B are formulas, and α is a program, then $A \vee B$, $\neg A$, and $\langle\alpha\rangle A$ are formulas.
(3) Every a_i is a program.
(4) If α, β are programs and A is a formula, then $\alpha \cup \beta$, $\alpha \cap \beta$, $\alpha; \beta$, α^*, and $A?$ are programs.

The semantics interprets formulas over models $\langle S, \pi, \rho\rangle$, where S is a set of states, π attaches a subset $\pi(P)$ of S to every atomic formula P, and ρ interprets atomic programs a.

In regular PDL, a program α is interpreted by a subset $\rho(\alpha)$ of $S \times S$, with $(s, s') \in \rho(\alpha)$ meaning that α can be executed from s to reach s'. In contrast, interpretation of programs in CPDL is based on the notion of *reachability pairs* of the form (s, U), where $s \in S$ and $U \subseteq S$. The semantics attaches a subset $\rho(a)$ of $S \times 2^S$ (henceforth, a *reachability set*) to every atomic program a. Intuitively, $(s, U) \in \rho(\alpha)$ for a reachability pair (s, U) and a program α means that α can be executed from s, in parallel, to reach all states of U. Thus, the formula $\langle\alpha\rangle A$ holds in a state s iff there is a set $U \subseteq S$ such that $(s, U) \in \rho(\alpha)$, and each state in U satisfies A.

Define the binary operator $\cdot$ over the domain $S \times 2^S$ by

$$R_1 \cdot R_2 = \{(s, U) \mid \exists s_1, U_1, s_2, U_2, \ldots, ((s, \{s_1, s_2, \ldots\}) \in R_1 \\ \wedge \forall i (s_i, U_i) \in R_2 \wedge U = \textstyle\bigcup_i U_i)\},$$

for all reachability sets $R_1, R_2 \subseteq S \times 2^S$.

For a given reachability set T, define the operator F_T over the domain $S \times 2^S$ by

$$F_T(R) = R_{true} \cup T \cdot R,$$

for every reachability set $R \subseteq S \times 2^S$, where

$$R_{true} = \{(s, \{s\}) \mid s \in S\}.$$

(This notation reflects the fact that by later definitions R_{true} is precisely $\rho(true?)$, the interpretation of the program testing for *true*.)

Note that the operator $\cdot$ is monotone (in the lattice-theoretic sense), that is, $R_1 \subseteq R_2 \Rightarrow R_0 \cdot R_1 \subseteq R_0 \cdot R_2$. Therefore F_T is also monotone for every T. This ensures that, for every T, the least fixpoint of F_T exists, by the Knaster-Tarski theorem [37]. $LFP(F_T)$ can be computed using the equivalent definition as the limit of the following sequence of "partial solutions": $R_0 = R_{true}$, $R_{i+1} = F_T(R_i)$ $(= R_{true} \cup T \cdot R_i)$, and $R_\lambda = \bigcup_{\gamma<\lambda} R_\gamma$, for a limit ordinal λ.

We extend π and ρ to all formulas and programs by the following rules:

$$\pi(A \vee B) = \pi(A) \cup \pi(B),$$
$$\pi(\neg A) = S - \pi(A),$$
$$\pi(\langle\alpha\rangle A) = \{s \mid \exists U((s, U) \in \rho(\alpha) \text{ and } U \subseteq \pi(A))\},$$

and

$$\begin{aligned}
\rho(A?) &= \{(s, \{s\}) \mid s \in \pi(A)\},\\
\rho(\alpha \cup \beta) &= \rho(\alpha) \cup \rho(\beta),\\
\rho(\alpha \cap \beta) &= \{(s, U) \mid \exists V, W((s, V) \in \rho(\alpha), (s, W) \in \rho(\beta) \text{ and } U = V \cup W)\},\\
\rho(\alpha; \beta) &= \rho(\alpha) \cdot \rho(\beta),\\
\rho(\alpha^*) &= LFP(F_{\rho(\alpha)}).
\end{aligned}$$

Note that the rule $\rho(\alpha^*) = \bigcup_i \rho(\alpha)^i$ $(= \bigcup_i \rho(\alpha^i))$ does not hold anymore in general. Intuitively, such a rule would force all branches of any parallel computation of $(\alpha \cap \beta)^*$ to be of the same depth.

Let us define also the dual connective $[\alpha]A$ as $\neg\langle\alpha\rangle\neg A$. The interpretation of the "box" connective is given by $\pi([\alpha]A) = \{ s \mid \forall V((s, V) \in \rho(\alpha) \Rightarrow V \cap \pi(A) \neq \emptyset)\}$. Thus $[\alpha]A$ states that in every possible execution of α, at least one of the end states satisfies A.

A reachability set $R \subseteq S \times 2^S$ is *finitely branched* iff for every s and U, $(s, U) \in R \Rightarrow |U|$ is finite. (Note that this notion refers only to the "and" branching of R. In contrast, finite "or" branching may be defined as meaning that for every s, the number of sets U such that $(s, U) \in R$ is finite.)

R is *sequential* iff for every s and U, $(s, U) \in R \Rightarrow |U| = 1$.

A *program* α is finitely branched or sequential iff $\rho(\alpha)$ is. A *model* is finitely branched or sequential iff every atomic program is interpreted in it as such.

LEMMA 2.1. *For all finitely branched reachability sets T and R, $T \cdot R$ and $LFP(F_T)$ are finitely branched.*

PROOF. The case of $T \cdot R$ is immediate from the definitions. The case of $LFP(F_T)$ is handled by simple transfinite induction on the sets R_i generated in the computation. □

LEMMA 2.2. *In a finitely branched model, every program is finitely branched.*

PROOF. By induction on the structure of programs. The case of α^* is handled by Lemma 2.1. □

LEMMA 2.3. *If T is a finitely branched reachability set, then F_T is continuous; that is, for any infinite chain $R_0 \subseteq R_1 \subseteq \cdots$ of reachability sets, $F_T(\bigcup_{i<\omega} R_i) = \bigcup_{i<\omega} F_T(R_i)$.*

PROOF. Let T and $R_0 \subseteq R_1 \subseteq \cdots$ be as in the lemma. Clearly, the $\supseteq$ inclusion is guaranteed by the monotonicity of F_T. Conversely, assume $(s, U) \in F_T(\bigcup_{i<\omega} R_i)$. The case of $(s, U) \in R_{true}$ is trivial. Otherwise, there are sets $V = \{s_1, \ldots, s_n\}$, $U_1, \ldots, U_n \subseteq S$ such that V is finite, $(s, V) \in T$, $U = \bigcup_{1 \le j \le n} U_j$, and for every $1 \le j \le n$, $(s_j, U_j) \in \bigcup_{i<\omega} R_i$. Therefore, for every $1 \le j \le n$ there is some i_j such that $(s_j, U_j) \in R_{i_j}$. Choose $k = \max\{i_1, \ldots, i_n\}$, and then for every $1 \le j \le n$, $(s_j, U_j) \in R_k$. Hence, $(s, U) \in F_T(R_k)$, so also $(s, U) \in \bigcup_{i<\omega} F_T(R_i)$. □

Note that continuity may be lost if T is infinitely branched. For example, let $S = \{j \mid 0 \le j < \omega\}$, $T = \{(j, S) \mid 0 \le j < \omega\}$, and $R_i = \{(j, \{k\}) \mid 0 \le j < \omega, 0 \le k \le i\}$, for every $0 \le i < \omega$. Then $F_T(\bigcup_i R_i)$ contains the pair (j, S) (for any j), whereas $\bigcup_i F_T(R_i)$ does not.

Hence, by the last two lemmas, if all atomic programs involved in a program α are finitely branched, then $\rho(\alpha^*)$ always converges within ω steps. (In contrast, an infinitely branched α may require more than ω steps to converge (M. Y. Vardi, private communication).) This property is not needed for most of the results in what follows. However, for compatibility, expressiveness comparisons with other logics will be carried over sequential models (which are, in particular, finitely branched).

Example 2.1. Considering models in the form of infinite full binary a/b trees, the formula *P-cut* addresses exactly those models in which there is a cut in the tree such that all the nodes of the cut satisfy P.

$$P\text{-}cut: \langle (a \cap b)^* \rangle\, P.$$

Alternatively, $\neg P$-*cut* (or, $[(a \cap b)^*]\neg P$) means that in every possible execution of $(a \cap b)^*$, at least one of the end states does not satisfy P. Put another way, $\neg P$-*cut* asserts the existence of an infinite a/b-path of states satisfying $\neg P$; thus over these models $\neg P$-*cut* is equivalent to **repeat**$((a \cup b); \neg P?)$ (**repeat**(α) is true in a state s if α can be repeatedly executed infinitely often; cf. [8]). We suspect that this is inexpressible in regular PDL (though our proof that CPDL > PDL uses a different example).

Let us now say a word on the description of *runs* of programs in CPDL. Since a specific run of a PDL program is fully deterministic, it can be described by a *computation sequence*, or seq (cf. [8]). Such a seq is a program of the form $\gamma_1; \cdots; \gamma_n$, where each γ_i is an atomic operation. In fact, we may replace a program α with $\bigcup_{\beta \in \alpha} \beta$, where the union is over all seqs β of α (meaning that β represents some run of α). However, seqs do not suffice to describe a *concurrent* run, such as that of a CPDL program. Such a run is actually in the form of a *tree computation*, whose branches represent the different parallel paths of the run (as for ATMs). Consequently, the corresponding deterministic program, called a *trec*, is a program obtained by inductively applying ";" and "$\cap$" to atomic programs. Here, too, we can identify a program α with a set $\mathcal{T}(\alpha)$ of trecs, such that α is equivalent to $\bigcup_{\gamma \in \mathcal{T}(\alpha)} \gamma$.

2.2 CPDL VERSUS GAME LOGIC. *Game logic* was recently introduced by Parikh [27, 28] as a propositional level logic for handling games and related issues. The work described in this paper was to a large extent triggered by Parikh's work and owes much to his ideas. In expressiveness, game logic is stronger than PDL and no more expressive than Kozen's propositional μ-calculus L_μ [19]. Game logic differs from CPDL mainly in the following points:

(1) In game logic, a *monotonicity* requirement is imposed on programs: $(s, U) \in \rho(\alpha)$ and $U \subseteq V$ implies $(s, V) \in \rho(\alpha)$. This requirement helps to simplify the semantic rules (especially when regarding $\rho(\alpha)$ as an operator on subsets of S; cf. [28]). (Note that this is not the same definition used before, although the underlying notion of monotonicity is identical.)
(2) In game logic, a general *dual* operator is added to programs. This operator is responsible for most of the additional (expressive) power of game logic and enables, for example, expressing **repeat**(α) for some given program α (cf., [27]).

Parikh [27, 28] concentrates on the game-oriented presentation of this formalism. Thus, (s, U) is interpreted as a set of moves from a state s, and the dual of α, α^d, is viewed as the same game with the two players interchanged. The monotonicity requirement also makes sense in this view. The logic in this formalism indeed enables a convenient manipulation of assertions regarding games, especially when coupled with the axiom system proposed by Parikh in [28] (though not known yet to be complete).

For our purposes, however, we are mainly interested in preserving the basic interpretation of dynamic logic as a program-oriented logic. This motivation leads us to drop the monotonicity restriction on $\rho(\alpha)$ in certain cases, since it does not correspond to the intuitive meaning of α's as concurrent programs; assuming $(s, U) \in \rho(\alpha)$, meaning that α can be executed from s, in parallel, to reach all states of U, we may not necessarily want α to be able to reach any larger set of states. In fact, we may want some programs to have deterministic, sequential semantics. (Note that this argument is limited to the issue of the naturalness and appropriateness of presentation. As for validity and expressability issues, we discuss the effects of monotonicity later.)

A more significant difference concerns the extent of "dualism" allowed in the logics. Game logic admits a dual for any of its connectives. In fact, as mentioned in [28], it is possible to replace the single dual operator α^d on composite programs with a set of dual connectives, one for each basic connective of DL (in addition to the dual of atomic programs, a^d). In particular, we may define $\alpha \cap \beta$ to replace $(\alpha^d \cup \beta^d)^d$, and α^∞ to replace $((\alpha^d)^*)^d$. $(\alpha^d;\ \beta^d)^d$ is simply $\alpha;\beta$, so no special new connective is necessary. On the other hand, CPDL allows only $\cap$, the dual of $\cup$.

The important point is that the introduction of the concurrency operator $\cap$ alone to DL does not change the original interpretation of $\langle\alpha\rangle A$ as "α can be executed so that *upon termination*, A holds" (assuming atomic programs are continuous, in particular finitely branched). This is because the convergence ordinals of programs remain no larger than ω. In contrast, the addition of other duals (in particular α^∞ and a^d for atomic a) may result in the loss of that meaning, since the inductive definition of $\rho(\alpha)$ for some programs α may require more than ω steps to converge. In fact, the α^∞ connective may be considered as a *repetition connective*, since the formula $\langle((\alpha^d)^*)^d\rangle$ *true* is equivalent to **repeat**(α). Thus, on the first-order level, for instance, we obtain a logic that is no longer a sublanguage of even QDL$_{\text{re}}$, which allows only recursively enumerable sets of finite computation paths as programs (cf. [12] and [23]). Since our primary goal is to accommodate concurrency, we find that it is not really necessary to strengthen the logic in such a way (which, as mentioned earlier, is also reflected in the issue of axiomatization).

Before proceeding to prove our main results, we would like to consider in more detail the requirement of monotonicity adopted in game logic. As argued before, monotonicity seems to obscure the intuitive meaning of programs as such and thus is inappropriate from a descriptional point of view. On a technical level, however, this dichotomy is not as severe as it may seem at first. Let us first consider a monotone semantics for tests. That is, define ρ_m just as ρ, except that $\rho_m(A?) = \{(s, U) \mid s \in U \cap \pi(A)\}$, and define π_m accordingly. (Here, $\rho(\mathit{true}?) \cdot R = R \cdot \rho(\mathit{true}?) = R$ for any *monotone* $R \subseteq S \times 2^S$, in the above sense). We then have

LEMMA 2.4. *For any program* α,

(1) $\rho(\alpha) \subseteq \rho_m(\alpha)$,
(2) *for any* $(s, U) \in \rho_m(\alpha)$ *there is a* $U' \subseteq U$ *such that* $(s, U') \in \rho(\alpha)$.

PROOF. By induction on the structure of α. Most cases are straightforward. Let us first illustrate the case of $\alpha = \beta;\gamma$. Claim (1) is obvious. As for (2), assume $(s, U) \in \rho_m(\alpha)$. Then there are $V = \{s_1, s_2, \ldots\}$, U_1, $U_2, \ldots$ such that $(s, V) \in \rho_m(\alpha)$ and $(s_i, U_i) \in \rho_m(\beta)$, and $U = \bigcup_i U_i$. By inductive hypothesis there are

$V' = \{s_{i1}, s_{i2}, \ldots\} \subseteq V$, $U'_1 \subseteq U_1$, $U'_2 \subseteq U_2, \ldots$ such that $(s, V') \in \rho(\alpha)$ and for any i_j, $(s_{i_j}, U'_{i_j}) \in \rho(\beta)$. Choose $U' = \bigcup_{i_j} U'_{i_j}$, and then $(s, U') \in \rho(\alpha;\beta)$.

The only somewhat involved case is that of $\alpha = \beta^*$. One can prove the claims by an internal (transfinite) induction on the sequence $R_0, R_1, \ldots$ of temporary solutions leading to the least fixpoint of the (monotone) equation $R = \rho(true?) \cup \rho(\beta) \cdot$ R. Details are omitted. □

LEMMA 2.5. *For any formula A and any model, $\pi_m(A) = \pi(A)$.*

PROOF. By induction on the structure of A. The nontrivial case is $A = \langle\alpha\rangle B$. To show that $\pi(A) \subseteq \pi_m(A)$, let $s \in \pi(A)$. There is a U such that $(s, U) \in \rho(\alpha)$ and $U \subseteq \pi(B)$. By part (1) of Lemma 2.4 $(s, U) \in \rho_m(\alpha)$, and by the inductive hypothesis $U \subseteq \pi_m(B)$, so $s \in \pi_m(A)$. Conversely, assume $s \in \pi_m(A)$. Then there exists a U such that $(s, U) \in \rho_m(\alpha)$ and $U \subseteq \pi_m(B)$. Now use part (2) of Lemma 2.4 to deduce the existence of $U' \subseteq U$ such that $(s, U') \in \rho(\alpha)$, and since, by the inductive hypothesis, we have $U' \subseteq \pi(B)$, we conclude that $s \in \pi(A)$. □

From this observation it is clear that for our purposes there is no difference between monotone and nonmonotone semantics for tests.

Now consider the restriction of CPDL to *monotone models*, that is, models in which $\rho(a)$ is monotone for each atomic program a. We make the following observations:

LEMMA 2.6. *Over monotone models, for any program α,*

(1) *$\rho_m(\alpha)$ is monotone,*
(2) *$\rho(\alpha) = \rho^1(\alpha) \cup \rho^2(\alpha)$, where $\rho^1(\alpha)$ is monotone and $\rho^2(\alpha)$ is a subset of $\rho(true?)$ (i.e., $\{(s, U) \mid s \in U\}$).*

PROOF. Both cases are easily proved by structural induction on programs. □

As a consequence of Lemma 2.5 we use the two options for tests interchangeably, preferring the nonmonotone semantics for expressability results and comparisons, and the monotone semantics for decidability and completeness. We do not require *models*, or atomic programs, to be monotone.

One final issue to be discussed is that of *sequentiality*. We would like to stress the point that the notion of sequentiality is of importance in game logic too, even though games, rather than programs, are what one has in mind; in actual games it is normally the case that moves take us from one state to another *single* state.

At first sight it may seem that the idea of sequentiality does not fit game logic, since it is in direct contradiction to the monotonicity requirement. Still, we can incorporate both properties by adopting the notion of *inherent sequentiality*; a program α is inherently sequential iff $(s, U) \in \rho(\alpha) \Rightarrow \exists t \in U((s, \{t\}) \in \rho(\alpha))$.

The usefulness of this notion stems from the fact that restriction to sequential models may also give rise to new valid formulas. For instance, $\langle\alpha\rangle A \vee \langle\alpha\rangle B \equiv \langle\alpha\rangle(A \vee B)$ is valid for a sequential (or inherently sequential) program α, whereas for a general α only the $\supset$ direction holds. Yet, the full equivalence is necessary in order to prove some very natural properties of games. For example, it is a well-known fact that there always exists a winning strategy for one of the players in finite Nim-like games. However, the formula expressing this fact in game logic is not generally valid. Indeed, we can prove this formula if we allow a primitive formula *sequential* with the obvious meaning, and assume an axiom *sequential*$(\alpha) \supset (\langle\alpha\rangle A \vee \langle\alpha\rangle B \equiv \langle\alpha\rangle(A \vee B))$. Actually, this axiom subsumes the entire difference between the sequential and the nonsequential interpretations, as implied by the remark at the end of Section 2.4.

2.3 CPDL VERSUS PDL AND THE μ-CALCULUS. In this section we consider only sequential CPDL models, defined above. Consequently, for any PDL model $\mathcal{M}$, there is a corresponding CPDL model $\mathcal{M}'$, with the same S and π, and where $(s, t) \in \rho(a)$ in $\mathcal{M}$ iff $(s, \{t\}) \in \rho(a)$ in $\mathcal{M}'$. In the sequel we identify $\mathcal{M}$ and $\mathcal{M}'$. For any two logics L_1 and L_2 interpreted over such PDL models, $L_1 \leq L_2$ means that for every formula $A \in L_1$ there is a formula $B \in L_2$ such that for every PDL model $\mathcal{M}$, $\pi(A) = \pi(B)$. $L_1 = L_2$ iff both $L_1 \leq L_2$ and $L_2 \leq L_1$. $L_1 < L_2$ iff $L_1 \leq L_2$ and $L_1 \neq L_2$.

First we compare CPDL and PDL in expressive power. It is obvious that CPDL $\geq$ PDL. Strictness is shown by a method based on an idea of Kozen [19]. Consider the following model M. It consists of an infinite sequence of states labeled 0, 1, 2, ... From every state i, $i > 0$ there are both a and b edges leading to $i - 1$. In addition, there is a "bypassing" edge emanating from every odd state $2i + 1$, $i > 0$ and reaching $2i - 1$. These edges are marked as follows: the first is an a-edge, the next is a b-edge, then come two a-edges and two b-edges, then three of each type, etc. (see Figure 1).

Consider the following CPDL formula:

$$\varphi: \langle((a \cap b); a)^*\rangle([a]\mathit{false}).$$

Clearly φ is satisfiable in every even state of the model. However, no odd state satisfies it, since in any possible run of the program $((a \cap b); a)^*$ from an odd state, some branch will always be forced to remain on the main path and will not use bypasses. This process will therefore not be able to reach state 0. Hence, $\pi(\varphi) = \{2i \mid i \geq 0\}$, or φ describes precisely the even states of the model. On the other hand, we have the following lemma, which is proved in the Appendix.

LEMMA 2.7. *For every PDL formula ψ, the set $\pi(\psi)$ in the model M is either finite or cofinite.*

This means that no PDL formula can be equivalent to φ. Thus

THEOREM 2.8. *CPDL > PDL.*

The additional expressive power of CPDL stems from the capability of spawning an unbounded number of processes in parallel (e.g., using programs of the form $(\alpha \cap \beta)^*$). Indeed, we may consider also a version with bounded degree of concurrency, that is, one in which $\cap$ is not allowed to occur under $*$. Then this version can easily be shown no more expressive than PDL.

Next we discuss relationships with the μ-calculus. Propositional μ-calculus, L_μ, defined by Kozen in [19], can be viewed as another extension of PDL. Its syntax contains atomic formulas P_i, atomic programs a_i, and the following construction rules:

(1) Each P_i is a formula.
(2) If A, B are formulas, a is an atomic program, and $F(X)$ is a formula with only positive occurrences of a new atomic symbol X, then $A \wedge B$, $A \vee B$, $\neg A$, $\langle a\rangle A$ and $\mu X.F(X)$ are formulas.

The semantics interprets formulas over models $\langle S, \pi, \rho\rangle$, where S is a set of states, π attaches a subset $\pi(P)$ of S to every atomic formula P, and ρ attaches a subset $\rho(a)$ of $S \times S$ to every atomic program a. We extend π to every formula by

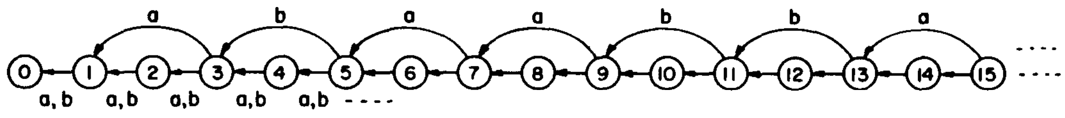

FIG. 1. The Model M.

the following rules:

$$\begin{aligned}\pi(A \vee B) &= \pi(A) \cup \pi(B),\\ \pi(\neg A) &= S - \pi(A),\\ \pi(\langle a\rangle A) &= \{s \mid \exists t((s, t) \in \rho(a) \text{ and } t \in \pi(A))\},\\ \pi(\mu X.F(X)) &= \min\{U \mid U \subseteq S \text{ and } U = F(U)\},\end{aligned}$$

where $F(U)$ stands for $\pi(F(X))$ with $\pi(X) = U$.

It is known [27] that L_μ subsumes the full game logic, but this containment is not known to be strict (though it has been shown that a certain extension of game logic is equivalent to L_μ (M. Y. Vardi, private correspondence)). It is also clear that L_μ subsumes CPDL. We now want to characterize sublanguages of L_μ, on the basis of certain restrictions, that give closer bounds to the power of CPDL.

Consider the *continuous* sublanguage of L_μ, named CL_μ, obtained by the additional requirement that in each formula $\mu X.F(X)$, appearances of X are under no negations at all. (This definition parallels that of Park [29] for first-order continuous μ-calculus. Pratt [34] defined a similar propositional version, with a seemingly weaker condition of continuity. It is possible, though, that the systems are actually equivalent in expressive power.)

Let us further restrict the language by adding a requirement we call *simplicity*. This means that interleavings of different μ's (e.g., formulas of the form $\mu X(\cdots \mu Y(\cdots X \cdots) \cdots))$ are prohibited. Call the resulting language *simple continuous* propositional μ-calculus, or SCL_μ.

It is shown later that on the first-order level, the continuous μ-calculus is equivalent to CDL. On the propositional level, however, we can show only that CPDL lies between CL_μ and its simple sublanguage, SCL_μ.

As shown in [19] and [34], it is actually possible to express formulas with *composite* programs in the μ-calculus, by defining the abbreviations

$$\begin{aligned}\langle\alpha \cup \beta\rangle A &\equiv \langle\alpha\rangle A \vee \langle\beta\rangle A,\\ \langle\alpha \cap \beta\rangle A &\equiv \langle\alpha\rangle A \wedge \langle\beta\rangle A,\\ \langle\alpha;\beta\rangle A &\equiv \langle\alpha\rangle(\langle\beta\rangle A),\\ \langle B?\rangle A &\equiv B \wedge A,\end{aligned}$$

and, finally,

$$\langle\alpha^*\rangle A \equiv \mu X.(A \vee \langle\alpha\rangle X),$$

where X is a new atomic symbol. Note also that $[\alpha]A$, or $\neg\langle\alpha\rangle\neg A$ in general, is allowed too.

The above discussion yields

LEMMA 2.9. *CPDL* $\leq$ CL_μ.

A formula of SCL_μ is said to be *X-free* if it contains no occurrences of X. It is said to be in *canonical form* if it contains no subformulas of the general form $\mu X.F(X)$, except within the abbreviation for α^*.

LEMMA 2.10. *For any formula in SCL_μ, there is an equivalent formula in canonical form.*

PROOF. By induction on the structure of a formula A. The interesting case is $A = \mu X.F(X)$, where, by the inductive hypothesis, $F(X)$ is assumed to be in canonical form. Hence, $F(X)$ actually consists of combinations of the form $\langle a\rangle G(X)$, $G(X) \vee H(X)$, $G(X) \wedge H(X)$, X, and X-free formulas, where $G(X)$ and $H(X)$ are in canonical form; recall that any subformula of the form $\neg B$ is X-free by the continuity assumption, and any subformula of the form $\mu Y.G(Y)$ (abbreviated as $\langle \beta^*\rangle B$ for some β and B), is X-free by the simplicity assumption.

We handle this case by transforming $F(X)$ into an equivalent canonical form $\langle\alpha\rangle X \vee Q$, where Q and α are X-free. The desired canonical equivalent for $\mu X.F(X)$ is then taken to be simply $\langle\alpha^*\rangle Q$.

The existence of the equivalent $\langle\alpha\rangle X \vee Q$ is shown by induction on the structure of $F(X)$. The base cases are those in which $F(X)$ is one of the following:

An X-free Q: Choose $\langle \mathit{false}?\rangle X \vee Q$.
A parameter X: Choose $\langle \mathit{true}?\rangle X \vee \mathit{false}$.

Now assume, by inductive hypothesis, the existence of equivalent $\langle\alpha_G\rangle X \vee Q_G$ and $\langle\alpha_H\rangle X \vee Q_H$ for $G(X)$ and $H(X)$, respectively. The term chosen in each case is as follows:

For $G(X) \vee H(X)$: $\langle\alpha_G \cup \alpha_H\rangle X \vee (Q_G \vee Q_H)$.
For $G(X) \wedge H(X)$: $\langle(\alpha_G \cap \alpha_H) \cup (Q_H?;\alpha_G) \cup (Q_G?;\alpha_H)\rangle X \vee (Q_G \wedge Q_H)$.
For $\langle a\rangle G(X)$: $\langle a;\alpha_G\rangle X \vee \langle a\rangle Q_G$.

(Recall that for a *sequential* program a, $\langle a\rangle A \vee \langle a\rangle B \equiv \langle a\rangle(A \vee B)$ is valid, even though for general programs only the $\supset$ direction holds.) □

THEOREM 2.11. $SCL_\mu \leq CPDL \leq CL_\mu$.

PROOF. Immediate from Lemmas 2.9 and 2.10. □

The exact condition that needs to be imposed on L_μ in order to make it equivalent to CPDL by means of the specific translations described above is rather complex and involves some conjunctivity and aconjunctivity requirements on subformulas, and we do not state it here. Nevertheless, we cannot exclude the possibility that all these versions are in fact equivalent.

2.4 COMPLETENESS AND DECIDABILITY OF CPDL.

Consider the following axiom system (A) for CPDL.

Axiom Schemes:

(A1) All tautologies of the propositional calculus.
(A2) $\langle\alpha;\beta\rangle A \equiv \langle\alpha\rangle\langle\beta\rangle A$.
(A3) $\langle\alpha \cup \beta\rangle A \equiv \langle\alpha\rangle A \vee \langle\beta\rangle A$.
(A4) $\langle\alpha \cap \beta\rangle A \equiv \langle\alpha\rangle A \wedge \langle\beta\rangle A$.
(A5) $\langle\alpha^*\rangle A \equiv A \vee \langle\alpha\rangle \langle\alpha^*\rangle A$.
(A6) $\langle A?\rangle B \equiv A \wedge B$.

Inference Rules:

$$\frac{A,\ A \supset B}{B},\qquad \frac{A \supset B}{\langle\alpha\rangle A \supset \langle\alpha\rangle B},\qquad \frac{\langle\alpha\rangle A \supset A}{\langle\alpha^*\rangle A \supset A}.$$

These are, actually, the axioms for "dual-free" game logic [28], with the addition of a new axiom scheme (A4) for the concurrency operator. Our axioms hold over all CPDL models, including the nonmonotone ones.

For technical reasons, it is more convenient to prove completeness assuming a monotone semantics ρ_m for tests. This makes no essential difference, as explained above.

Our proof of completeness follows that of [28] for "dual-free" game logic, and we therefore only sketch it here. The Fischer–Ladner closure [5] of a formula A_0 is constructed much in the same way, with the addition of the following construction rule: If $\langle\alpha \cap \beta\rangle A \in FL(A_0)$, then also $\langle\alpha\rangle A$, $\langle\beta\rangle A \in FL(A_0)$. The model for A_0 is then defined similarly. Note that this model happens to be monotone, so by Lemma 2.6, $\rho_m(\alpha)$ is monotone for any α. Therefore, the rest of the proof follows directly that of [28], with minor changes, mainly the addition of the appropriate "$\alpha \cap \beta$ cases" in the inductive proofs of Lemmas 2 and 3 therein. Thus,

LEMMA 2.12. *Every consistent formula of CPDL has a "small" monotone model. (Here "small" means double exponential in the size of the formula.)*

THEOREM 2.13. *The axiom system (A) is complete for CPDL.*

Remark. In [31], we consider the version of CPDL restricted to *sequential* models, with a nonmonotone (sequential) semantics for tests, and prove completeness for the same axiom system, augmented with

(A7) $\langle a\rangle(A \vee B) \equiv \langle a\rangle A \vee \langle a\rangle B$ for an atomic program a.

The proof is slightly more involved, but again it essentially uses the ideas of the proof in [28].

The linear reduction of CPDL to CL_μ (Lemma 2.9), coupled with the fact that CL_μ is decidable in deterministic exponential time ((M. Y. Vardi, private correspondence); cf. [42]), yields

THEOREM 2.14. *The validity problem for CPDL can be decided in exponential time.*

We also have a matching lower bound for the validity problem, which is the same bound given in [5], for regular PDL, since CPDL is an extension of PDL.

THEOREM 2.15. *There is a constant $c > 0$ such that the validity problem for CPDL is not solvable in deterministic time 2^{cn}, where n is the length of the input formula.*

Hence, the validity problem of CPDL is in deterministic exponential time and is tight up to a polynomial (in the length of formulas) in the exponent.

3. *The First-Order Level*

3.1 CONCURRENT QDL. CQDL relates to QDL just as CPDL relates to PDL. The syntax is that of QDL, with the addition of the concurrency operator $\cap$ on programs.

The semantics is based on a first-order structure, and employs a set S of states, which are determined—as in QDL—by the values of all involved variables. A subset $\pi(\varphi)$ of S is attached to every formula φ, and a subset $\rho(\alpha)$ of $S \times 2^S$ to every program α. The values of $\pi(\varphi)$ are determined just as in ordinary QDL, with the obvious modification

$$\pi(\langle\alpha\rangle\varphi) = \{ s \mid \exists V((s, v) \in \rho(\alpha) \wedge V \subseteq \pi(\varphi))\},$$

and ρ is defined just as in CPDL, with an additional rule for assignments:

$$\rho(x \leftarrow t) = \{(s, \{s\,|^{x}_{t'}\}) \mid t' \text{ is the evaluation of the term } t \text{ in } s\}.$$

3.2 CQDL VERSUS QDL, THE μ-CALCULUS, QDL$_{\text{proc}}$, AND PTIME. We start by collecting some necessary preliminaries. The first-order μ-calculus Q$_\mu$, defined in [15], is an extension of first-order logic (with equality) by allowing formulas of the form $\mu R.F(R)$, where R occurs only positively in F. We omit a more thorough description and merely note that positivity of R in F ensures that $F(R)$, as a function over relations, is monotone and hence that it has a unique least fixpoint. A stronger syntactic requirement is that in $\mu R.F(R)$ the symbol R does not appear under negations at all (so F is continuous as a function). The resulting language is denoted CQ$_\mu$.

We also describe certain versions of QDL. QDL$_{\text{proc}}$ is obtained from QDL by enriching the control structure of programs with recursive procedures; the recursive program α^X may include an atomic command X, interpreted as "call α". Recursive calls are allowed to employ parameters (in the "call by reference" sense). Actually, we define a slightly different, yet equivalent, mechanism, in which local variables, rather than parameters, are added. A recursive program with local variables is denoted $\alpha(\bar{x})^X$, where the variables of $\bar{x}$ are global; that is, their value may change during the execution of a recursive call. All other variables are local; that is, they are pushed onto a stack upon entering a recursive call and restored upon its completion.

In order to avoid ambiguities, we require each recursive procedure α to have its own distinct "call" symbol X_α. It is also required that a call X_α appear only within the procedure α itself.

Another system we refer to is L$_{\omega_1\omega}$. L$_{\omega_1\omega}$ is an extension of first-order logic (FO), which allows also countable disjunctions of formulas. We particularly consider two sublanguages of L$_{\omega_1\omega}$. The one is *constructive* L$_{\omega_1\omega}$, or L$^{CK}_{\omega_1\omega}$, which allows only disjunctions of recursively enumerable sequences of formulas. The other is *bounded memory* L$_{\omega_1\omega}$, or L$_{\text{BM}}$, defined in [38]. For a term σ, let $rc(\sigma)$ be the minimal number of variables necessary to compute σ by a sequence of *basic* assignments; that is, $x \leftarrow y_1$ or $x \leftarrow f(y_1, \ldots, y_k)$, where $x, y_1, \ldots, y_k$ are variables. Given an L$_{\omega_1\omega}$ formula φ, $rc(\varphi)$ denotes $sup\{rc(\sigma) \mid \sigma \text{ is a term in } \varphi\}$. The language L$_{\text{BM}}$ is the collection of all L$_{\omega_1\omega}$ formulas φ with finite quantifier depth and $rc(\varphi) < \omega$.

The following proposition is a result of Tiuryn [38].

PROPOSITION 3.1 [38]. *$QDL_{\text{proc}} \not\leq L_{BM}$.*

We also define the language **ran-QDL$_{\text{proc}}$**, where **ran-L** denotes L enriched with a random assignment operation, $x \leftarrow ?$.

PROPOSITION 3.2 [10]. *CQ_μ =* **ran-**QDL_{proc}.

We also discuss the following special operation. Consider the structure $\mathscr{A} = \langle D, f_1, \ldots, f_l, c_1, \ldots, c_m, P_1, \ldots, P_k\rangle$ with domain D, functions $f_1, \ldots, f_l$, constants $c_1, \ldots, c_m$, and predicates $P_1, \ldots, P_k$. Let $\bar{x} = (x_1, \ldots, x_n)$ be a tuple of free variables. Define the *Herbrand universe* $HU(\bar{x})$ as the subset of D consisting of all elements equivalent to terms of the language over $\bar{x}$ (i.e., reachable from $c_1, \ldots, c_m, x_1, \ldots, x_n$ by means of $f_1, \ldots, f_l$). Define *Hran*$(\bar{x}, z)$ as a program assigning to z an arbitrary element of $HU(\bar{x})$. Denote by **Hran**-L the dynamic logic L enriched with the *Hran* operation.

It is clear that *Hran* is programmable in $\mathrm{QDL}_{\mathrm{proc}}$. As we shall see, the inherent incapability of programming *Hran* in CQDL renders this language strictly less expressive.

We are also interested in drawing connections between CQDL and PTIME. For this purpose we rely on the works of Immerman and Vardi, relating logics and complexity classes [16, 40]. The basic framework for this work involves finite structures containing a total ordering relation on the domain. Denote the collection of such structures by $(F, \leq)$. For a logic L, a complexity class C, and a collection W of structures, we say that $\mathrm{L} =_W \mathrm{C}$ iff (1) for every closed formula $\Phi \in \mathrm{L}$, $MOD_W(\Phi) \in \mathrm{C}$, and (2) for every set $S \subseteq W$, if $S \in \mathrm{C}$, then there is a formula $\Phi \in \mathrm{L}$ such that $MOD_W(\Phi) = S$ (where $MOD_W(\Phi)$ is the set of structures from W that satisfy Φ).

To relate finite structures to complexity classes, we assume a standard encoding of functions and relations as input tapes for Turing machines.

PROPOSITION 3.3 [16, 40]. $CQ_\mu =_{(F,\leq)} PTIME$.

Now we are ready to discuss the relationships between CQDL and the above-mentioned systems. Since the structures over which CQDL is defined are the same as for $\mathrm{QDL}_{\mathrm{proc}}$ and the μ-calculus, it is possible to compare these logics in expressive power.

LEMMA 3.4. $CQDL \leq L_{BM}$.

PROOF. Given a CQDL formula φ, we construct an $\mathrm{L_{BM}}$ equivalent inductively. The only nontrivial case is $\langle\alpha\rangle\varphi$. This formula is first transformed into $\langle\alpha;\varphi?\rangle$*true.* By the inductive hypothesis there are $\mathrm{L_{BM}}$ equivalents for φ and for every test in α. Let $\mathcal{T}(\alpha;\varphi?)$ be the set of trecs of $\alpha;\varphi?$ (so that $\rho(\alpha;\varphi?) = \bigcup_{\beta\in\mathcal{T}(\alpha;\varphi?)}\rho(\beta)$).

Now, for every trec β there is an $\mathrm{L_{BM}}$ formula ψ_β equivalent to $\langle\beta\rangle$*true.* This formula can be constructed from $\langle\beta\rangle$*true* by inductively applying the following transformation rules, whenever possible, and then replacing every test with its $\mathrm{L_{BM}}$ equivalent.

Trec Transformation Rules:

$$\begin{aligned}\langle\beta;\beta'\rangle\zeta &\Rightarrow \langle\beta\rangle(\langle\beta'\rangle\zeta),\\ \langle\beta\cap\beta'\rangle\zeta &\Rightarrow \langle\beta\rangle\zeta \wedge \langle\beta'\rangle\zeta,\\ \langle\zeta'?\rangle\zeta &\Rightarrow \zeta' \wedge \zeta,\\ \langle x \leftarrow \sigma\rangle\zeta &\Rightarrow \zeta[\sigma/x],\end{aligned}$$

where $\zeta[\sigma/x]$ is the usual substitution of σ for x in ζ.

Finally, $\bigvee_{\beta\in\mathcal{T}(\alpha;\varphi?)}\psi_\beta$ is the $\mathrm{L_{BM}}$ equivalent of $\langle\alpha\rangle\varphi$.

To see that this $\mathrm{L}_{\omega_1\omega}$ formula is really in $\mathrm{L_{BM}}$, note that the original formula is finite; so it contains only a finite number of terms and its quantifier depth is bounded. Since terms are left unchanged during the translation and the quantifier depth does not increase, these properties are preserved in the resulting formula ψ. □

The following is the main result of this section:

THEOREM 3.5

(1) $CQDL < QDL_{\mathbf{proc}}$,
(2) **ran**-$CQDL$ = **ran**-$QDL_{\mathbf{proc}}$,
(3) **Hran**-$CQDL = QDL_{\mathbf{proc}}$.

Consequently, from Propositions 3.2 and 3.3 we have, respectively,

COROLLARY 3.6. **ran**-$CQDL = CQ_{\mu}$.

COROLLARY 3.7. **ran**-$CQDL =_{(F,\le)} PTIME$.

Also note that CQDL < **ran**-CQDL. We still do not know (though we conjecture) that CQDL is strictly more expressive than QDL. However, for the restricted version of QDL^{qf}, which allows only first-order, quantifier-free tests in programs, we can show

THEOREM 3.8. $QDL^{\text{qf}} < CQDL^{\text{qf}}$.

PROOF. The proof relies strongly on a recent result of Urzyczyn [39]. Consider the structures of the form $\mathscr{A} = \langle \mathscr{D}, f, R, T, F\rangle$, where $\mathscr{D} = \{(i, j) \mid i = 0, 1; j = 1, \dots, n\}$ is the domain, f is a unary function defined by

$$f(i, j) = \begin{cases} (i, j+1), & i = 0 \text{ and } j < n, \\ (i, j), & \text{otherwise,} \end{cases}$$

the binary relation R is a subset of $\{((0, i), (1, i)) \mid 1 \le i \le n\}$, and $F = (0, 1)$, $T = (1, 1)$ are constants. Such a structure satisfies the property *parity* if R contains an even number of tuples. Figure 2 describes a structure satisfying *parity* (with $n = 5$).

Urzyczyn [39] has shown that *parity* is not expressible in QDL^{qf}. We now show how to construct a formula expressing *parity* in CQDL^{qf}. We first construct a program whose job is to spawn n processes (denoted, for the moment, by $P_1, \dots, P_n$). Each process P_i maintains a variable x holding $(0, i)$, and two additional variables, *arrow* and *even*, holding F or T. The formula *parity* then claims that it is possible to execute this program so that in the end, in each process P_i, *arrow* = T iff R contains $((0, i), (1, i))$, and *even* = T iff the number of tuples $((0, j), (1, j))$ in R such that $j \le i$ is even, and in addition, *even* = T in the last process P_n.

Define the following programs:

$$\begin{aligned}
&\textit{init}: && y \leftarrow T; x \leftarrow F; (\textit{arrow} \leftarrow T;\ \textit{even} \leftarrow F \cup \textit{arrow} \leftarrow F; \textit{even} \leftarrow T),\\
&\textit{flip}: && \textit{even} = T?; \textit{even} \leftarrow F \cup \textit{even} = F?; \textit{even} \leftarrow T,\\
&\textit{spawn}: && (y = T?; x \ne f(x)?; ((\textit{arrow} \leftarrow F; y \leftarrow F \cap x \leftarrow f(x))\\
& && \qquad \cup (\textit{arrow} \leftarrow T; y \leftarrow F \cap x \leftarrow f(x);\ \textit{flip})))^{*}.
\end{aligned}$$

Finally, the desired formula is

$$\begin{aligned}
\textit{parity}: \langle \textit{init}; \textit{spawn}\rangle((y = T \supset (x = f(x) \wedge \textit{even} = T))\\
\wedge\ \textit{arrow} = T \equiv \exists z R(x, z)). \qquad \square
\end{aligned}$$

Note. Urzyczyn's proof that *parity* is inexpressible in QDL^{qf} fails for CQDL^{qf} in a very early stage, for the following reason. The proof assumes (without loss of generality) that any QDL^{qf} formula is written in prenex form, that is, as $Q_1 x_1 \cdots Q_k x_k \Psi$, where Ψ is an open formula. This assumption is no longer valid in CQDL;

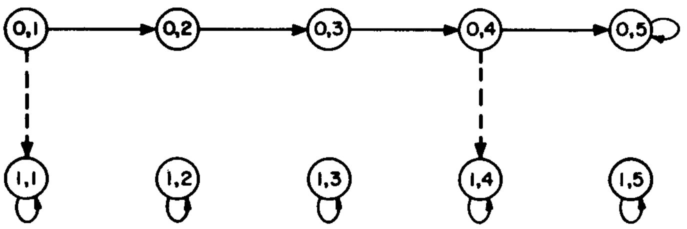

FIG. 2. A structure satisfying *parity* ($n = 5$). Solid arrows denote f, and broken ones denote R.

formulas containing programs that spawn an unbounded number of processes (i.e., contain a subprogram $(\alpha \cap \beta)^*$) cannot in general be "prenexed." In contrast, formulas that use only a bounded number of processes, have equivalent formulas in QDL, and this is true even in the general case of rich tests. Hence, just as in the propositional case, the version of CQDL with bounded concurrency is no more expressive than QDL.

PROOF OF THEOREM 3.5

Part (1). We only have to show that CQDL $\le$ $\mathrm{QDL}_{\mathrm{proc}}$; then strictness follows from Proposition 3.1 and Lemma 3.4.

Given a formula of CQDL, we first rewrite subformulas of the form $\langle\alpha\rangle A$ as $\langle\alpha;A?\rangle true$. We now show how to construct an equivalent formula in $\mathrm{QDL}_{\mathrm{proc}}$ for formulas of the form $\langle\alpha\rangle true$ in CQDL.

The idea of the simulation is best explained by means of an example. Suppose we want to translate $\langle\alpha \cap \beta\rangle A$, or $\langle(\alpha \cap \beta);A?\rangle true$, where $\bar{x}$ is the tuple of free (input) variables of $\alpha \cap \beta$, and let $B \in \mathrm{QDL}_{\mathrm{proc}}$ be the equivalent of A. The related computation tree of Figure 3 is simulated in $\mathrm{QDL}_{\mathrm{proc}}$ by a program that activates $(\alpha;B?)$ and $(\beta;B?)$ as recursive calls, while saving $\bar{x}$ on a stack when entering a recursive call, and retrieving it upon termination.

To make this more formal, define $sub(\alpha)$ as the collection of subprograms of α, and attach a label l_β to every $\beta \in sub(\alpha)$; also define l_{end} to denote the end of the program. For example, given $\alpha = (((a;b) \cup c)^*)$, with subprograms $\beta = ((a;b) \cup c)$, $\gamma = (a;b)$, etc., label subprograms by $l_\alpha = 1$, $l_\beta = 2$, $l_\gamma = 3$, $l_c = 4$, $l_a = 5$, $l_b = 6$, and $l_{\mathrm{end}} = 7$.

Next, assign a set N_β of labels to every $\beta \in sub(\alpha)$. The labels of N_β are those to which one can proceed after executing β. These sets are defined by induction, as follows:

$\beta = \alpha$ (*the entire program*): $N_\alpha = \{l_{\mathrm{end}}\}$.
$\beta = \gamma \cup \delta$: N_γ, $N_\delta = N_\beta$.
$\beta = \gamma \cap \delta$: N_γ, $N_\delta = N_\beta$.
$\beta = \gamma;\delta$: $N_\gamma = \{l_\delta\}$.
$N_\delta = N_\beta$.
$\beta = \gamma^*$: $N_\gamma = \{l_\beta\} \cup N_\beta$.

Next, construct a term β' for each β in $sub(\alpha)$ by induction on the structure of α. β' simulates β using recursion instead of concurrency; concurrent execution of $\gamma \cap \delta$ is simulated by two recursive calls, performed sequentially. Since both branches have to execute independently, starting with the same initial values in mutual variables, it is necessary to save these values for the second branch, using

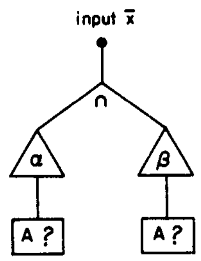

FIG. 3. Computation tree of $\langle(\alpha \cap \beta); A?\rangle$ *true*, to be simulated in $\mathrm{QDL_{proc}}$.

the mechanism of local variables. It is also necessary to keep track of the current label. This does not require the existence of integers in the domain; rather, because program texts are finite, it can be handled by a tuple of variables $\bar{z}$ holding binary representations of labels, provided that there are at least two distinct elements in the universe (i.e., $\exists x \exists y(x \neq y)$ holds). The case of a singleton universe is left to the reader.

Atomic β (test, assignment) (Tests $A?$ are assumed to have been translated properly into $\mathrm{QDL_{proc}}$ beforehand):

$$\text{let } \beta' = (\bar{z} = l_\beta)?;\beta;(\textstyle\bigcup_{l \in N_\beta} \bar{z} \leftarrow l);X,$$
$$\beta = \gamma \cup \delta\text{: let } \beta' = (\bar{z} = l_\beta)?;(\bar{z} \leftarrow l_\gamma \cup \bar{z} \leftarrow l_\delta);X,$$
$$\beta = \gamma;\delta\text{: let } \beta' = (\bar{z} = l_\beta)?;\bar{z} \leftarrow l_\gamma;X,$$
$$\beta = \gamma \cap \delta\text{: let } \beta' = (\bar{z} = l_\beta)?;\bar{m} \leftarrow \bar{x};\bar{z} \leftarrow l_\gamma;X;\bar{x} \leftarrow \bar{m};\bar{z} \leftarrow l_\delta;X$$

where $\bar{x}$ is the tuple of free variables of α, and $\bar{m}$ contains new *local* variables. Here is where we make essential use of the feature of local variables, in order to store values during a recursive call.

$$\beta = \gamma^*\text{: let } \beta' = (\bar{z} = l_\beta)?;(\bar{z} \leftarrow l_\gamma \cup \textstyle\bigcup_{l \in N_\beta} \bar{z} \leftarrow l);X.$$

In addition let $\beta_{\text{end}} = (\bar{z} = l_{\text{end}})?$.

Finally, the formula corresponding to $\langle\alpha\rangle$*true* is taken to be

$$\forall \bar{b} \langle \bar{z} \leftarrow l_\alpha;(\beta_{\text{end}} \cup \textstyle\bigcup_{\beta \in sub(\alpha)} \beta')^X\rangle true,$$

where $\bar{b}$ is the tuple of all new local variables added during the construction.

Part (2). The ($\leq$) direction is similar to that of Part (1). We have to show that **ran-CQDL** $\geq$ **ran-QDL**$_{\mathbf{proc}}$. Thus we wish to construct an equivalent formula in **ran-CQDL** for $\langle\alpha\rangle$*true* $\in$ **ran-QDL**$_{\mathbf{proc}}$. Again we use an example in order to clarify the idea of the simulation. Suppose we want to translate the formula

$$\langle(\alpha_1 \cup \alpha_2;X_\alpha;\alpha_3 \cup \alpha_4;(\beta_1 \cup \beta_2;X_\alpha;\beta_3 \cup \beta_4;X_\beta;\beta_5;X_\alpha)(\bar{y})^{X_\beta};\alpha_5)(\bar{x})^{X_\alpha}\rangle A,$$

or, actually, to simulate the program

$$(\alpha_1 \cup \alpha_2;X_\alpha;\alpha_3 \cup \alpha_4;(\beta_1 \cup \beta_2;X_\alpha;\beta_3 \cup \beta_4;X_\beta;\beta_5;X_\alpha)(\bar{y})^{X_\beta};\alpha_5)(\bar{x})^{X_\alpha};A?.$$

Later we refer to the two recursive procedures of this program as α and β. A possible run is described in Figure 4. The corresponding computation tree in **ran-CQDL** should take the form of Figure 5.

The general idea of the simulation is as follows. The flow of control is handled just as in the converse simulation, on the basis of keeping track of the current label and using a "next command" function. Now suppose a process reaches a recursive call X_α (say, in the $\cdots \alpha_2;X_\alpha;\alpha_3 \cdots$ in the example). At this point, the following

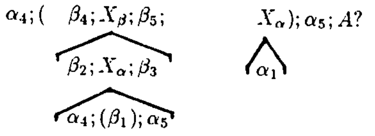
$\alpha_4;(\quad \beta_4;X_\beta;\beta_5;$
$X_\alpha);\alpha_5;A?$
$\beta_2;X_\alpha;\beta_3$
α_1
$\alpha_4;(\beta_1);\alpha_5$

FIG. 4. A possible run of the recursive program in the example.

has to be done. First we have to "guess" (using random assignments) the values of the global variables $\bar{x}$ upon returning from the recursive call. Now we split into two parallel processes. The "left" one saves the guessed values in special variables $\bar{x}_{\text{save}}$ and turns to execute the recursive call (i.e., execute α again). At the end of its main path, it verifies that the final values of the global variables are identical to the guessed values and then terminates. The "right" process assumes the guessed values to be correct, assigns them into the global variables, and proceeds as if "just returned" from the recursive call.

Some technical problems arise in implementing this idea. To begin with, there is a need to identify the "endpoint" of a recursive procedure. This is done by attaching to the end of each recursive procedure $\alpha(\bar{x})^{X_\alpha}$ a dummy atomic program fin_α (say, $x_1 \leftarrow x_1$), thus turning it into $(\alpha; fin_\alpha)(\bar{x})^{X_\alpha}$. Our example program turns into

$$((\alpha_1 \cup \alpha_2;X_\alpha;\alpha_3$$
$$\cup\ \alpha_4;((\beta_1 \cup \beta_2;X_\alpha;\beta_3 \cup \beta_4;X_\beta;\beta_5;X_\alpha);fin_\beta)(\bar{y})^{X_\beta};\alpha_5);fin_\alpha)(\bar{x})^{X_\alpha};A?.$$

Now consider this program and assume that at some moment a process finishes β_3 and reaches fin_β. It is now important for the process to remember how it entered β. If it entered β directly (i.e., after finishing α_4), then it has to proceed in α and execute α_5. On the other hand, if the process has initiated β owing to a recursive call X_β, then it is a left branch; so it merely has to verify the correctness of the guess made at the point of the call and then terminate. Thus we need a mechanism to distinguish between being in the highest level and being in some internal level. This is done by employing a variable $level_\beta$ for every recursive procedure β, being 1 when in the highest level (i.e., when entering β regularly through the flow of the program) and 0 when in an internal level (i.e., when invoked by a recursive call).

At first, it seems that several consecutive entries of alternating nature into β might destroy our trace of levels, since we use a single variable (rather than a stack). To see that no such problem arises, one notes that once inside β, the only way to enter it again in a regular (flow) entry is to execute a recursive call X_α first, in which case the right branch of the split will safely maintain the correct value of $level_\beta$ for the outer β. This situation is demonstrated in the example (Figures 4 and 5).

We proceed to give a precise description of the simulation. Again define $sub(\alpha)$ as the collection of subprograms of α, and attach a label l_β to every $\beta \in sub(\alpha)$; in particular, each occurrence of X_γ receives a label. Also define l_{end} to denote the end of the program.

Next, assign a set N_β of labels to every $\beta \in sub(\alpha)$. The labels in N_β are those to which one can proceed after executing β. These sets are defined as in Part (1), omitting the cases of $\cap$ and $*$ and adding $N_\gamma = N_\beta$ for $\beta = \gamma^{X_\gamma}$.

Next, construct a term β' for each β in $sub(\alpha)$, by induction on the structure of α. The term β' simulates β as explained. Keeping track of the current label is done just as in the previous simulation.

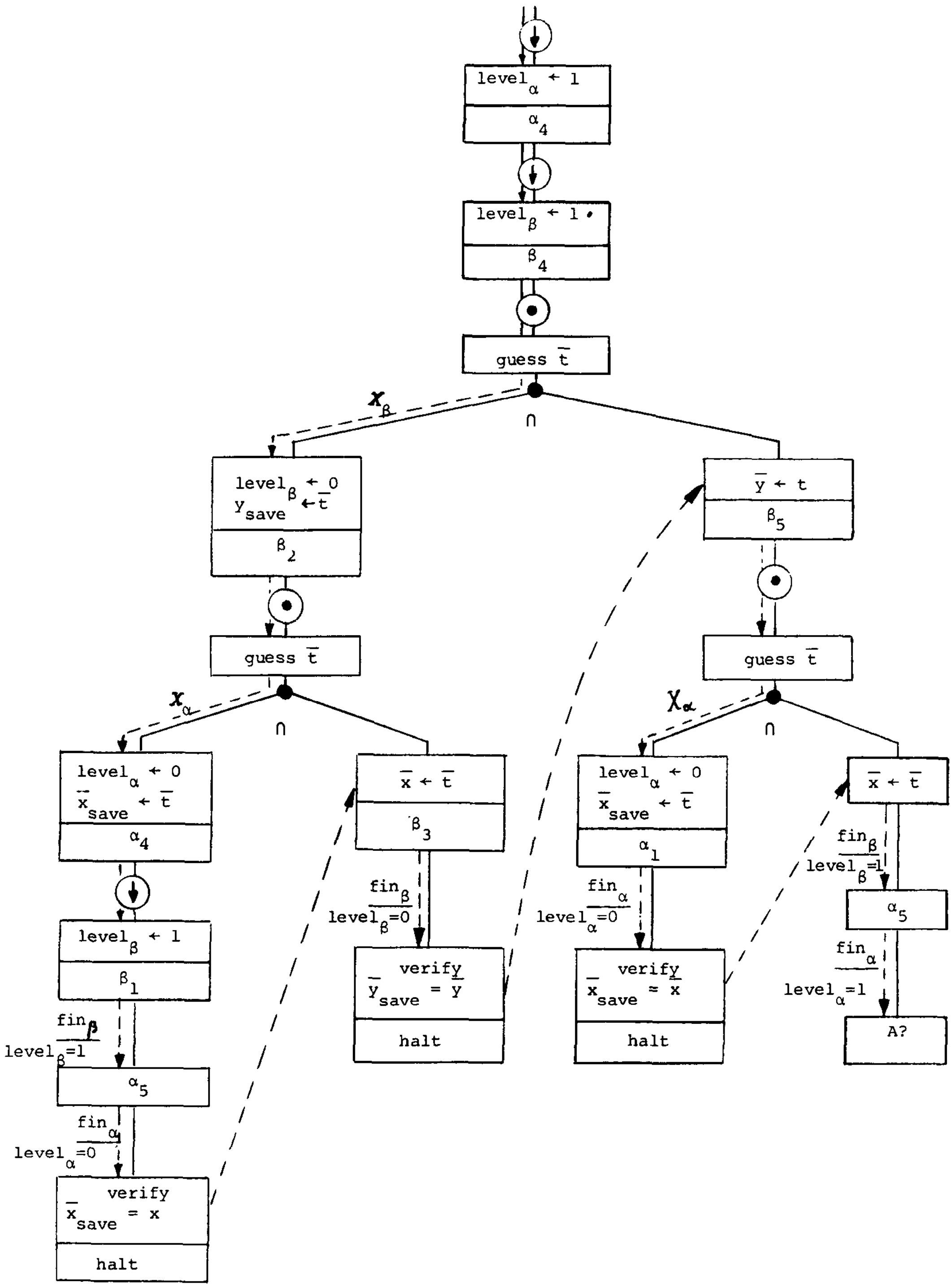

FIG. 5. ran-CQDL computation tree simulating the **proc** program run of Figure 4. ⊙ denotes points of entering a recursive procedure by a recursive call (in the original program), while (↓) denotes regular entries into the procedure. Broken lines trace the simulated run on the tree.

Atomic β (*test, simple/random assignment*): (Tests $A?$ are assumed to have been translated properly into **ran**-CQDL beforehand).

$$\begin{array}{ll}
 & \text{let } \beta' = (\bar z = l_\beta)?;\beta;(\bigcup_{l\in N_\beta} \bar z \leftarrow l).\\
\beta=\gamma\cup\delta: & \text{let } \beta' = (\bar z = l_\beta)?;(\bar z \leftarrow l_\gamma \cup \bar z \leftarrow l_\delta).\\
\beta=\gamma;\delta: & \text{let } \beta' = (\bar z = l_\beta)?;\bar z \leftarrow l_\gamma.\\
\beta=\gamma^{X_\gamma}: & \text{let } \beta' = (\bar z = l_\beta)?;(\bar z \leftarrow l_\gamma).\\
\beta=X_\gamma: & \text{let } \beta' = (\bar z = l_\beta)?;\bar t \leftarrow ?;\\
 & \qquad ((\bar z \leftarrow l_\gamma;\bar x^\gamma_{\text{save}} \leftarrow \bar t;\mathit{level}_\gamma \leftarrow 0)\\
 & \qquad \cap (\bigcup_{l\in N_\beta} \bar z \leftarrow l;\bar x^\gamma \leftarrow \bar t)).\\
\beta=\mathit{fin}_\gamma: & \text{let } \beta' = (\bar z = l_\beta)?;\\
 & \qquad ((\mathit{level}_\gamma = 0?;\bar x^\gamma_{\text{save}} = \bar x^\gamma?;\mathit{halt})\\
 & \qquad \cup (\mathit{level}_\gamma = 1?;\bigcup_{l\in N_\beta} \bar z \leftarrow l)).
\end{array}$$

In the last three cases, X_γ is the "call" symbol of $\gamma(\bar x^\gamma)^{X_\gamma}$, where $\bar x^\gamma$ is the tuple of variables global in (the original) γ, the tuples $\bar x^\gamma_{\text{save}}$ and $\bar t$ contain new variables, level_γ is a new variable, and fin_γ is the dummy program indicating the end of γ. The command *halt* appearing in the last case is interpreted as $\rho(\mathit{halt}) = \{(s, \emptyset) \mid s \in S\}$, so a "halted" branch is never considered afterward.

Finally, the formula corresponding to $\langle\alpha\rangle\mathit{true}$ is taken to be

$$\forall \bar b \langle \bar z \leftarrow l_\alpha;\mathit{level}_\alpha \leftarrow 1;(\bigcup_{\beta\in sub(\alpha)}\beta')^*\rangle \bar z = l_{\text{end}},$$

where $\bar b$ is the tuple of all new variables added during the construction.

The *halt* command can now be eliminated inductively as follows: Given $\langle\alpha\rangle A$ (where A contains no appearances of *halt*, by the inductive hypothesis), construct the formula

$$\langle b \leftarrow 1;\alpha'\rangle(b = 0 \vee (b = 1 \wedge A)),$$

where b is a new variable, and α' is obtained from α by replacing every subprogram β with $b = 0? \cup (b = 1?;\beta)$, and every *halt* command with $b \leftarrow 0$.

Part (3). The ($\le$) direction is similar to that of Part (1), relying on the fact that *Hran* is programmable in QDL_{proc}. The ($\ge$) direction is similar to that of Part (2); note that random assignments were used in the simulating program only to guess values in the Herbrand universe (in fact, values that were later reached directly through the course of computation), so *Hran* could be used equally well. □

3.3 Validity and Axiomatization.

Validity of CQDL formulas is clearly Π^1_1-complete, since

$$\text{QDL} \le \text{CQDL} \le \text{QDL}_{\text{re}} \le \text{L}^{CK}_{\omega_1\omega},$$

where the translations are all effective and the logics on both extremes have a Π^1_1-complete validity problem [14, 17].

As for the issues of axiomatization, we follow the results regarding QDL, as summarized in [8]. In fact, all axiom systems and completeness results therein carry over to the concurrent version with minor changes. The proofs too are appropriate adaptations of those described in [8]. Thus in the uninterpreted case we can give an axiom system (B) which completely axiomatizes termination assertions, and an infinitary axiom system (C), (i.e., with an infinitary inference rule) that completely axiomatizes CQDL.

In the interpreted case, we restrict ourselves to expressive or arithmetical structures (cf. [8]), and consider axiom systems that are complete relative to any given structure $\mathscr{A}$. Here we can give an axiom system (D), which is relatively complete for partial correctness assertions over expressive structures, and an axiom system (E) containing an induction rule, which is relatively complete for CQDL over arithmetical structures or is arithmetically complete.

We give the results without proofs, as they are natural extensions of the corresponding results in [8]. An exception is the infinitary axiom system C, whose completeness for QDL is only sketched in [8]; we give the full proof for the concurrent version CQDL in the Appendix.

Uninterpreted Case

Complete Axiomatization for Termination Assertions. First we consider only termination assertions, of the form $\varphi \supset \langle\alpha\rangle\psi$. Consider the following axiom system (B).

Axiom Schemes:

(B1) All instances of valid CPDL formulas.
(B2) All valid first-order formulas.
(B3) $\varphi[\sigma/x] \supset \langle x \leftarrow \sigma\rangle\varphi$, for variable x, term σ and first-order φ.

Inference Rules:

$$\frac{\varphi,\ \varphi \supset \psi}{\psi}, \quad \frac{\varphi \supset \psi}{\langle\alpha\rangle\varphi \supset \langle\alpha\rangle\psi}, \quad \frac{\langle\alpha\rangle\varphi \supset \varphi}{\langle\alpha^*\rangle\varphi \supset \varphi}.$$

Denote by $\vdash_B \varphi$ the provability of φ in the system (B).

THEOREM 3.9 (cf. [8, 22]). *For any CQDL formula χ of the form $\varphi \supset \langle\alpha\rangle\psi$, for first-order φ and ψ and program α with first-order tests,* $\models \chi$ iff $\vdash_B \chi$.

General Infinitary Axiomatization. We follow the conventions of [8] regarding the definitions of free and bound variables of QDL formulas. The infinitary axiom system (C) we consider is the following:

Axiom Schemes:

(C1) All instances of valid CPDL formulas.
(C2) All instances of valid first-order formulas.
(C3) $\langle x \leftarrow \sigma\rangle\varphi \equiv \varphi[\sigma/x]$.

The application of (C2) requires us to give a precise definition for the notions of free and bound occurrences of variables within CQDL formulas. We assume some standard definition for these notions (cf. chap. 3.4.2 of [8]).

Inference Rules:

For every program α and formula ψ, let $\psi^0_{[\alpha]} = \psi$ and $\psi^{i+1}_{[\alpha]} = \psi^i_{[\alpha]} \wedge [\alpha]\psi^i_{[\alpha]}$ (for every $i \geq 0$), and dually let $\psi^0_{\langle\alpha\rangle} = \psi$ and $\psi^{i+1}_{\langle\alpha\rangle} = \psi^i_{\langle\alpha\rangle} \vee \langle\alpha\rangle\psi^i_{\langle\alpha\rangle}$ (for every $i \geq 0$). Note that for any $i \geq 0$, $\neg\psi^i_{\langle\alpha\rangle} \equiv (\neg\psi)^i_{[\alpha]}$. Now the rules are

$$\frac{\varphi,\ \varphi \supset \psi}{\psi}, \quad \frac{\varphi}{[\alpha]\varphi,\ \forall x\varphi}, \quad \frac{\varphi \supset \psi}{\langle\alpha\rangle\varphi \supset \langle\alpha\rangle\psi}, \quad \frac{\{\varphi \supset \psi^i_{[\alpha]}\}_{i<\omega}}{\varphi \supset [\alpha^*]\psi}.$$

The infinitary rule of CQDL is more complicated than that of QDL owing to the fact that in CDL, $\rho(\alpha^*)$ cannot be decomposed into $\bigcup_{i<\omega} \alpha^i$.

Note also that though the rule of monotonicity is derivable from the rule of generalization in regular DL, this is apparently no longer the case in CDL.

THEOREM 3.10 (cf. [8, 21]). *For any CQDL formula φ, $\models\varphi$ iff $\vdash_C \varphi$.*

PROOF. Left to the Appendix.

INTERPRETED CASE. We now restrict ourselves to expressive or arithmetical structures. Expressive structures are structures over which CQDL is reducible to FO, including, in particular, all arithmetical and all finite structures (cf. [8]). We consider axiom systems that are complete relative to any given expressive or arithmetical structure $\mathscr{A}$, that is, when given all $\mathscr{A}$-valid first-order formulas as axioms.

Relative Axiomatization for Correction Assertions. First we consider only partial correctness assertions, of the form $\varphi \supset [\alpha]\psi$ for first-order (or, program-free) φ, ψ. Consider the following axiom system (D).

Axiom Schemes:

(D1) All instances of valid CPDL formulas.
(D2) $\langle x \leftarrow \sigma\rangle\varphi \equiv \varphi[\sigma/x]$, for first-order φ.

Inference Rules:

$$\frac{\varphi,\ \varphi \supset \psi}{\psi}, \qquad \frac{\varphi \supset \psi}{\langle\alpha\rangle\varphi \supset \langle\alpha\rangle\psi}, \qquad \frac{\varphi}{[\alpha]\varphi}.$$

Given a structure $\mathscr{A}$, denote by $\mathscr{A} \vdash_D \varphi$ the provability of φ in the system (D) extended by the following set of axioms:

($D_{\mathscr{A}}$) All $\mathscr{A}$-valid first-order formulas.

THEOREM 3.11 (cf. [4, 8]). *For every expressive structure $\mathscr{A}$ and for every CQDL formula χ of the form $\varphi \supset [\alpha]\psi$, for first-order φ and ψ and program α with first-order tests, $\mathscr{A} \models \chi$ iff $\mathscr{A} \vdash_D \chi$.*

General Arithmetical Axiomatization. The general axiom system (E) we consider is

Axiom Schemes:

(E1) All instances of valid CPDL formulas.
(E2) All instances of valid first-order formulas.
(E3) $\langle x \leftarrow \sigma\rangle\varphi \equiv \varphi[\sigma/x]$, for first-order φ.

Inference Rules:

$$\frac{\varphi,\ \varphi \supset \psi}{\psi}, \qquad \frac{\varphi}{[\alpha]\varphi,\ \forall x\varphi}, \qquad \frac{\varphi \supset \psi}{\langle\alpha\rangle\varphi \supset \langle\alpha\rangle\psi},$$

and the induction rule

$$\frac{\varphi(n+1) \supset (\varphi(n) \vee \langle\alpha\rangle\varphi(n))}{\varphi(n) \supset \langle\alpha^*\rangle\varphi(0)}$$

for first-order φ, and n not appearing in α.

THEOREM 3.12 (*cf.* [8]). *For every arithmetical structure $\mathscr{A}$ and for every CQDL formula χ, $\mathscr{A} \models \chi$ iff $\mathscr{A} \vdash_E \chi$.*

Appendix A: Proof of Lemma 2.7

We have to show that for every PDL formula ψ, the set $\pi(\psi)$ in M is either finite or cofinite. For this we need some preliminary observations.

We distinguish in M between two sorts of edges, namely, *straight* and *bypassing* ones. The latter are further divided into *a-bypasses* and *b-bypasses.* A maximal sequence of states covered by adjacent a-bypasses (respectively, b-bypasses) is called an *a-interval* (respectively, *b-interval*). Denote the mth a-interval by A_m (respectively, B_m). Note that the length (i.e., number of states) of A_m (respectively, B_m) grows by 2 with every increase of m.

We need the following elementary facts:

LEMMA 1. *For any PDL programs* α, β, γ,

(1) $\rho(\alpha) \subseteq \rho(\alpha \cup \beta)$,
(2) $\rho(\alpha) \subseteq \rho(\alpha^*)$,
(3) $\rho((\alpha;\alpha)^*) \subseteq \rho(\alpha^*)$,
(4) $\rho(\alpha) \subseteq \rho(\beta) \Rightarrow \rho(\alpha;\gamma) \subseteq \rho(\beta;\gamma)$,
(5) $\rho(\alpha) \subseteq \rho(\beta) \Rightarrow \rho(\gamma;\alpha) \subseteq \rho(\gamma;\beta)$,
(6) $\rho(\alpha) \subseteq \rho(\beta) \Rightarrow \rho(\alpha \cup \gamma) \subseteq \rho(\beta \cup \gamma)$,
(7) $\rho(\alpha) \subseteq \rho(\beta) \Rightarrow \rho(\alpha^*) \subseteq \rho(\beta^*)$.

Given a program α and a state s in M, denote by $T(\alpha, s)$ the set of states i such that $(i, s) \in \rho(\alpha)$.

LEMMA 2. *For every state s in M, and every test-free program α^* containing only atomic a and b, $T(\alpha^*, s)$ is cofinite.*

PROOF. Construct β' from α by repeatedly replacing every subprogram $\gamma \cup \delta$, and every subprogram γ^* by γ. This yields eventually a sequence of a's and b's. Let $\beta = \beta';\beta'$. By Lemma 1, $\rho(\beta^*) \subseteq \rho(\alpha^*)$; thus it suffices to show that $T(\beta^*, s)$ is cofinite, where $\beta^* = (\gamma_1; \cdots; \gamma_n)^*$, $\gamma_i \in \{a, b\}$ for $1 \le i \le n$, and n is even.

Let s be given. Define $\xi(k, i) = s + kn + i$. Thus $\{\xi(k, i) \mid k \ge 0, n > i \ge 0\}$ is the set of states from s onward. We group these states in two different ways. The first is to divide the line from s onward into segments of n states each, denoted $\beta_k = (\xi(k, 0), \ldots, \xi(k, n-1))$. The other is to group all states in the ith position of some segment into $S_i = \{\xi(k, i) \mid k \ge 0\}$ (see Figure 6).

It is obvious that all states of S_0 belong to $T(\beta^*, s)$; from any state $\xi(k, 0) = s + kn$ it is possible to reach s by k *straight executions* of β, that is, executions through straight edges only. Such a straight execution passes through segments $\beta_{k-1}, \ldots, \beta_0$. Call the set S_0 the *basic execution set of* β^* *with respect to* s.

Now, if state t is in S_i, then a straight β-computation starting at t will terminate at $t - n$, which is another state of S_i. However, if it is possible to execute β using some available *bypassing* edge, then we may reach a state of S_{i-1}. Clearly, if it is possible to execute β^* starting in some $t \in S_i$ and ending in some state of S_0 (by using bypassing edges in some of the executions of β), then t is in $T(\beta^*, s)$. We intend to show that this is indeed the case for *any* state t, from some point on. For this purpose we show the existence of $n - 1$ *bypassing executions* of β, $(\xi(k_1 + 1, 1), \xi(k_1, 0)), \ldots, (\xi(k_{n-1} + 1, n-1), \xi(k_{n-1}, n-2)) \in \rho(\beta)$, where $k_{n-1} > \cdots > k_1$. (See Figure 7.) Then, for any state $t \ge \xi(k_{n-1} + 1, n-1)$, we have $(t, s) \in \rho(\beta^*)$.

Actually, we prove a stronger claim, namely, that for any i, $n > i \ge 0$, and any state l, $l \ge s$, there exists a k such that $\xi(k, i) > l$ and $(\xi(k+1, i+1), \xi(k, i)) \in \rho(\beta)$.

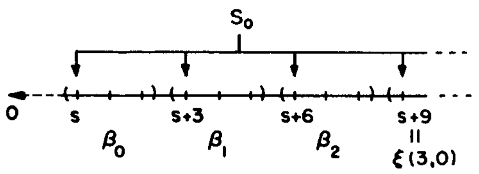

FIG. 6. The β_k-segments and the ith positions ($n = 3$).

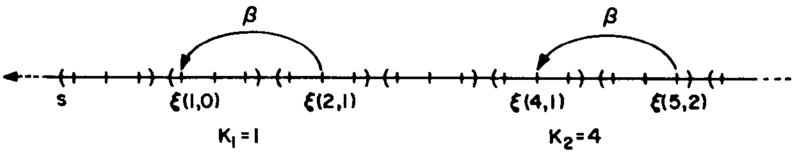

FIG. 7. Bypassing β-computations ($n = 3$).

Consider some given i, $n > i \geq 0$, and l, $l \geq s$. We choose to demonstrate a bypassing β-computation using a single bypassing edge, by one of the first two atomic components of β, $\gamma_1;\gamma_2$. Specifically, if $s + i$ is even, use γ_1, and otherwise use γ_2. We assume without loss of generality that $s + i$ is even, and that $\gamma_1 = a$. The other cases are treated similarly.

Recall that the length of a-intervals grows linearly with m. Hence for $m \geq \max(l, 2n)$, A_m contains at least $4n$ states; thus it contains two complete β-segments, β_k, β_{k+1}. (See Figure 8.)

For this particular k we claim that $(\xi(k + 1, i + 1), \xi(k, i)) \in \rho(\beta)$. To see this, note that $\xi(k + 1, i + 1) = (s + i) + (k + 1)n + 1$ is odd; hence, there is an a-bypass leaving it. Therefore, β can be executed from $\xi(k + 1, i + 1)$ by using this bypass for γ_1, and following straight edges for the rest of β, ending in $\xi(k, i)$. □

In the absence of tests, the rest of the proof follows easily from Lemma 2. It remains to be shown how to account also for programs *with* tests.

First we need some definitions concerning PDL programs and tests. Call a test $A?$ *finite* if $\pi(A)$ is finite in the model M. A program α is said to be *finite test free* if subprograms γ^* of α contain no occurrences of finite tests.

We also define a standard (disjunctive) form for PDL programs. A program α is in *standard form* if for any subprogram β^*, β is of the form $\bigcup_{1 \leq i \leq m} \gamma_i$, and each γ_i is of the form $\gamma_i = \delta_{i,1}; \cdots; \delta_{i,n_i}$ where each $\delta_{i,j}$ is either a test, an atomic program or some θ^* in standard form.

LEMMA 3. *For every PDL program α there is an equivalent program β in standard form* (*i.e.*, $\rho(\beta) = \rho(\alpha)$ *in every model*).

PROOF. The desired β can be obtained from α by repeatedly applying the following (valid) transformation rules to any subprogram of α, whenever possible:

(1) $(\delta \cup \epsilon);\gamma \Rightarrow \delta;\gamma \cup \epsilon;\gamma$,
(2) $\gamma;(\delta \cup \epsilon) \Rightarrow \gamma;\delta \cup \gamma;\epsilon$. □

A PDL program α is said to be *a/b real* if it is in standard form, and for any subprogram β^* of α, where $\beta = \bigcup_{1 \leq i \leq m} \gamma_i$, any γ_i contains some occurrences of a or b, that is, no seq consists solely of tests (though tests may occur in any seq). A PDL program α is in *separated form* if $\alpha = \alpha' \cup A?$ where α' is an a/b real program. (Either part may be totally omitted.)

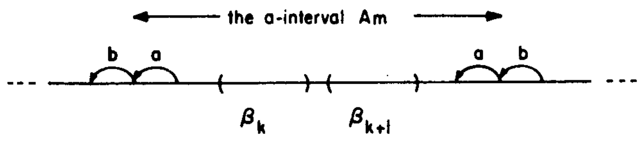

FIG. 8. The a-interval A_m.

LEMMA 4. *For every PDL program α containing no atomic programs other than a or b, there is an equivalent program $\alpha' \cup A?$ in separated form.*

PROOF. Construct α' and A inductively, as follows:

$a/b/A?$: Leave α as it is.

For the rest of the cases we assume $\beta = \beta' \cup B?$, $\gamma = \gamma' \cup C?$. If one part is missing, the treatment is changed accordingly.

$\beta \cup \gamma$: Let $\alpha' = \beta' \cup \gamma'$, and $A? = (B \vee C)?$.
$\beta;\gamma$: Let $\alpha' = \beta';C? \cup B?;\gamma' \cup \beta';\gamma'$, and $A? = (B \wedge C)?$.
β^*: Let $\alpha' = \beta';\beta'^*$, and $A? = true?$.

Next, apply steps of Lemma 3 once more to α', to regain standard form, if necessary. □

LEMMA 5. *For any PDL program α with no occurrences of atomic programs other than a or b, there is a finite-test-free program $\alpha' \cup A?$ in separated form, equivalent to α in M.*

PROOF. Assume α contains l different finite tests $A_1?, \ldots, A_l?$, and let $d_i = |\pi(A_i)|$ in M, for $1 \le i \le l$. Let $\alpha' \cup A?$ be a program in separated form equivalent to α. Now repeat the following process as long as α' is not finite test free. Let β^* be some most internal subprogram that is not finite test free; that is, $\beta = \bigcup_{1 \le i \le m} \gamma_i$ and some γ_k contains a finite test A_j on its highest level.

(1) Define the program $\epsilon = \bigcup_{\substack{1 \le i \le m \\ i \ne k}} \gamma_i$ (if $m = k = 1$ set $\epsilon = true?$).
(2) Replace β^* with $\bigcup_{0 \le i \le d_j} (\epsilon^*;(\gamma_k;\epsilon^*)^i)$.
(3) Transform the whole program again into separated form.

The final result of this process is the desired $\alpha' \cup A?$. To see that $\alpha' \cup A?$ is equivalent to α in M note that for any intermediate α'' containing some γ_k with occurrences of a finite $A_j?$ on its highest level, any trec of α'' containing *more* than d_j successive appearances of trecs of γ_k *must* fail; it has to test for A_j in more than d_j *distinct* states, owing to the acyclic nature of the model and the fact that γ_k is a/b real. □

Finally we are ready to prove Lemma 2.7 itself, namely, that in the model M, for every CPDL formula A, the set $\pi(A)$ is either finite or cofinite. Let A be given. Start by replacing all occurrences of atomic formulas P with *false*, and all occurrences of atomic programs c (other than a or b) with *false*?. This obviously yields an equivalent formula over M.

The claim is now proved by induction on the structure of A. The cases of *false*, *true*, negation and disjunction are immediate. We show the case of $\langle\alpha\rangle A$. By the inductive hypothesis, $\pi(B)$ is either finite or cofinite for any test $B?$ in α. By Lemma 5, replace α with a separated, finite-test-free version, equivalent to the original in M. If this separated form is just a test $B?$, then it must be either finite or cofinite (being a disjunctive/conjunctive closure of tests of α), and we are done, since $\langle\alpha\rangle A$ is equivalent to $B \wedge A$. Otherwise, we proceed by induction on the structure of α', the a/b real part of α, showing that for any formula B, if $\pi(B)$ is finite/cofinite, then so is $\pi(\langle\alpha'\rangle B)$. Again we omit the trivial cases of a, b, $\beta \cup \gamma$, $\beta \cap \gamma$, and $\beta;\gamma$, and consider the case of β^*. Recall that β is a real a/b program, and contains only cofinite tests. Let these tests be $A_1, \ldots, A_k?$. Since all of them are cofinite, there is a state l from which onward all these tests are true, that is, such that for any $j \geq l$, $j \in \pi(A_i)$, for any $1 \leq i \leq k$. Now there are two possibilities:

(1) For every state $s \geq l$, $s \notin \pi(\langle\beta^*\rangle A)$. In this case $\pi(\langle\beta^*\rangle A)$ is clearly finite, and we are done.

(2) There is a state $s \geq l$ such that $s \in \pi(\langle\beta^*\rangle A)$. In this case, construct from β^* a new program γ^* by (1) transforming β^* into standard form, (2) replacing any test $A_j?$ with *true*?, and (3) eliminating all appearances of *true*? by applying the following transformation rules:

$$\begin{aligned} \gamma;\mathit{true}? &\Rightarrow \gamma, \\ \mathit{true}?;\gamma &\Rightarrow \gamma, \\ (\gamma \cup \mathit{true}?)^* &\Rightarrow \gamma^*, \\ (\mathit{true}?)^* &\Rightarrow \mathit{true}?. \end{aligned}$$

The obtained γ^* contains no tests at all, and no atomic programs other than a and b. By Lemma 2, $T(\gamma^*, s)$ is cofinite, so there is some $t \geq s$ such that $(j, s) \in \rho(\gamma^*)$ for any $j \geq t$. But since $j \geq t \geq s \geq l$, it is clear that the computation path of γ^* from j to s passes only through states satisfying $A_1, \ldots, A_k$. Therefore, there is an identical computation path of β^*, so $(j, s) \in \rho(\beta^*)$. Hence, $T(\beta^*, s)$ is also cofinite, and since $s \in \pi(\langle\beta^*\rangle A)$, any state $j \in T(\beta^*, s)$ is also in $\pi(\langle\beta^*\rangle A)$; so $\pi(\langle\beta^*\rangle A)$ must be cofinite too. □

Appendix B: Proof of Theorem 3.10

Soundness is straightforward. Completeness is shown by extending to CQDL the completeness proof sketched in [8] for QDL (which in turn, is an adaptation of the proof given in [17] for $L_{\omega_1\omega}$). We need the following definitions and facts.

Given a finite set A of CQDL formulas, denote the conjunction $\bigwedge_{\varphi\in A} \varphi$ by $\hat{A}$. Call such a set A *consistent* if $\hat{A}$ is (i.e., $\nvdash_C \neg\hat{A}$). An *atom* is any finite consistent set A of CQDL formulas, possibly involving some elements from among a countable set G of new constant symbols.

Let φ^+ be a consistent closed CQDL formula. We have to show the existence of a model for φ^+.

Define CL to be the least set of CQDL formulas satisfying

(1) $\varphi^+ \in CL$,
(2) $\exists x\varphi \in CL \Rightarrow \varphi[\sigma/x] \in CL$, for every closed term σ,
(3) $\varphi_1 \vee \varphi_2 \in CL \Rightarrow \varphi_1, \varphi_2 \in CL$,
(4) $\neg\varphi_1 \in CL \Rightarrow \varphi_1 \in CL$,
(5) $\langle x \leftarrow \sigma\rangle\varphi \in CL \Rightarrow \varphi[\sigma/x] \in CL$,
(6) $\langle\psi?\rangle\varphi \in CL \Rightarrow \psi \wedge \varphi \in CL$,
(7) $\langle\alpha \cup \beta\rangle\varphi \in CL \Rightarrow \langle\alpha\rangle\varphi \vee \langle\beta\rangle\varphi \in CL$,
(8) $\langle\alpha;\beta\rangle\varphi \in CL \Rightarrow \langle\alpha\rangle(\langle\beta\rangle\varphi) \in CL$,
(9) $\langle\alpha \cap \beta\rangle\varphi \in CL \Rightarrow \langle\alpha\rangle\varphi \wedge \langle\beta\rangle\varphi \in CL$,
(10) $\langle\alpha^*\rangle\varphi \in CL \Rightarrow \varphi^i_{\langle\alpha\rangle} \in CL$ for every $i \geq 0$,

(11) $(\sigma_1 = \sigma_2) \in CL$, for every two closed terms σ_1 and σ_2,
(12) CL is closed under substitution, that is, $\varphi \in CL$, σ is a term in φ and $c \in G \Rightarrow \varphi[\sigma/c] \in CL$.

Assume some enumeration $\{\varphi_i \mid i \geq 0\}$ for CL, and some enumeration $\{\sigma_i \mid i \geq 0\}$ for the closed terms. Construct an infinite sequence of atoms A_i as follows. First let $A_0 = \{\varphi^+\}$. Now, A_{i+1} is constructed from A_i by adding some formulas to it, according to the following rules.

(1) Add the formula $\sigma_i = c$, for some new constant symbol c (i.e., not appearing in any formula of A_i).
(2) If $A_i \cup \{\varphi_i\}$ is inconsistent, add $\neg\varphi_i$.
(3) Otherwise, add φ_i, and also include some formulas according to the form of φ_i:
   (3.1) $\varphi_i = \exists x\psi$: Also include $\psi[c/x]$, where $c \in G$ is some new constant symbol.
   (3.2) $\varphi_i = \langle\alpha^*\rangle\psi$: Also include $\psi^j_{\langle\alpha\rangle}$ for some j such that $\hat{A}_i \wedge \psi^j_{\langle\alpha\rangle}$ is consistent.

(There must be some j with this property; otherwise, assuming $\vdash_C \neg(\hat{A}_i \wedge \psi^j_{\langle\alpha\rangle})$ or $\vdash_C \hat{A}_i \supset \neg\psi^j_{\langle\alpha\rangle}$ for every $j \geq 0$, we get $\{\vdash_C \hat{A}_i \supset (\neg\psi)^j_{[\alpha]}\}_{j<\omega}$; hence, by the infinitary rule $\vdash_C \hat{A}_i \supset [\alpha^*]\neg\psi$, or $\vdash_C \neg(\hat{A}_i \wedge \langle\alpha^*\rangle\, \psi)$, contradicting the consistency of $\hat{A}_i \cup \{\varphi_i\}$.)

Finally let $A_\infty = \bigcup_i A_i$. Note that every finite subset of A_∞ is consistent, and that for every $\varphi_i \in CL$, either φ_i or $\neg\varphi_i$ appears in A_∞.

Construct a model $\mathscr{A}$ as follows. Divide G into equivalence classes $\hat{c} = \{d \in G \mid (d = c) \in A_\infty\}$. Let the domain of $\mathscr{A}$ be $D = \{\hat{c} \mid c \in G\}$, interpret predicates by $P_{\mathscr{A}} = \{(\hat{c}_1, \ldots, \hat{c}_k) \mid P(c_1, \ldots, c_k) \in A_\infty\}$, identify constants $c \in G$ as $\hat{c}$, and interpret other functions by defining $f_{\mathscr{A}}(\hat{c}_1, \ldots, \hat{c}_k) = \hat{c}$ iff $(f(c_1, \ldots, c_k) = c) \in A_\infty$.

Lemma 1. *For every closed term σ and every $c \in G$, $(\sigma = c) \in A_\infty$ iff $\mathscr{A} \models (\sigma = c)$ (or equivalently $\sigma_{\mathscr{A}} = c$, where $\sigma_{\mathscr{A}}$ denotes the interpretation of σ in $\mathscr{A}$).*

Proof. By induction on the structure of the term σ.

d: Immediate by the construction of $\mathscr{A}$.

$f(\sigma_1, \ldots, \sigma_k)$: By the construction of A_∞, it includes some positive formulas $(\sigma_i = c_i)$, for $c_i \in G$, $1 \leq i \leq k$, and by the inductive hypothesis $\mathscr{A} \models (\sigma_i = c_i)$, for $1 \leq i \leq k$. Also, since CL is closed under substitutions, $(f(c_1, \ldots, c_k) = c) \in CL$, so it must appear either negatively or positively in A_∞. Now:

($\Rightarrow$): Assume $(f(\sigma_1, \ldots, \sigma_k) = c) \in A_\infty$. Then $f(c_1, \ldots, c_k) = c$ must appear positively in A_∞, since otherwise the subset

$$\{\neg(f(c_1, \ldots, c_k) = c),\ (f(\sigma_1, \ldots, \sigma_k) = c)\} \cup \{(\sigma_i = c_i) \mid 1 \leq i \leq k\}$$

of A_∞ is inconsistent by (C2). Hence by the construction of $\mathscr{A}$, $f_{\mathscr{A}}(\hat{c}_1, \ldots, \hat{c}_k) = \hat{c}$, so $\mathscr{A} \models (f(\sigma_1, \ldots, \sigma_k) = c)$.

($\Leftarrow$): Assume $\mathscr{A} \models (f(\sigma_1, \ldots, \sigma_k) = c)$. Then $f_{\mathscr{A}}(\hat{c}_1, \ldots, \hat{c}_k) = \hat{c}$. By the construction of $\mathscr{A}$, $(f(c_1, \ldots, c_k) = c) \in A_\infty$. This forces $f(\sigma_1, \ldots, \sigma_k) = c$ to appear positively in A_∞, since otherwise, by (C2) again, the subset

$$\{\neg(f(\sigma_1, \ldots, \sigma_k) = c),\ (f(c_1, \ldots, c_k) = c)\} \cup \{(\sigma_i = c_i) \mid 1 \leq i \leq k\}$$

of A_∞ is inconsistent. □

LEMMA 2. *For every formula* $\varphi \in CL$, $\mathscr{A} \models \varphi$ *iff* $\varphi \in A_\infty$.

PROOF. Define the well-ordering $\prec$ on CL by (the transitive closure of) the following clauses:

(1) $\varphi \prec \varphi \vee \psi$ and $\psi \prec \varphi \vee \psi$,
(2) $\varphi \prec \neg\varphi$,
(3) $\varphi[c/x] \prec \exists x\varphi$,
(4) $\varphi[\sigma/x] \prec \langle x \leftarrow \sigma\rangle\varphi$,
(5) $\psi \wedge \varphi \prec \langle\psi?\rangle\varphi$,
(6) $\langle\alpha\rangle\varphi \vee \langle\beta\rangle\varphi \prec \langle\alpha \cup \beta\rangle\varphi$,
(7) $\langle\alpha\rangle\varphi \wedge \langle\beta\rangle\varphi \prec \langle\alpha \cap \beta\rangle\varphi$,
(8) $\langle\alpha\rangle\langle\beta\rangle\varphi \prec \langle\alpha;\beta\rangle\varphi$,
(9) $\psi^i_{\langle\alpha\rangle} \prec \langle\alpha^*\rangle\psi$, for every $i \geq 0$.

We proceed to prove the claim by induction on the ordering $\prec$.

$\varphi = P(\sigma_1, \ldots, \sigma_k)$ (*including equality*, $\sigma_1 = \sigma_2$): By the construction of A_∞, it includes also some positive formulas $\sigma_i = c_i$, for every $1 \leq i \leq k$, and by Lemma 1, $\mathscr{A} \models (\sigma_i = c_i)$, so $\mathscr{A}$ interprets σ_i as $\hat{c}_i$, for every $1 \leq i \leq k$. Also, $P(c_1, \ldots, c_k) \in CL$, so it must appear either positively or negatively in A_∞.

Now both directions are handled just as in the case of $f(\sigma_1, \ldots, \sigma_k) = c$ in Lemma 1.

$\varphi = \exists x\psi$: Assume $\exists x\psi \in A_\infty$. Then by the construction of A_∞, $\psi[c/x]$ appears positively in A_∞ for some $c \in G$. By the inductive hypothesis, $\mathscr{A} \models \psi[c/x]$. Hence, $\mathscr{A} \models \exists x\psi$. Conversely, assume $\mathscr{A} \models \exists x\psi$. Then $\mathscr{A}$ interprets x as some element $\hat{c}$ in the domain. Thus, $\mathscr{A} \models \psi[c/x]$. By the inductive hypothesis $\psi[c/x] \in A_\infty$. Now $\exists x\psi \in CL$, so it must appear in A_∞, and this appearance must be positive, as $\{\forall x\neg\psi, \psi[c/x]\}$ is inconsistent by axiom (C2) (or specifically, by the axiom instance $\forall x\varphi \supset \varphi[c/x]$).

$\varphi = \psi_1 \vee \psi_2$: Assume $\psi_1 \vee \psi_2 \in A_\infty$. Then at least one of the ψ_j's (without loss of generality ψ_1) must appear positively in A_∞, since $\{\psi_1 \vee \psi_2, \neg\psi_1, \neg\psi_2\}$ is inconsistent. By the inductive hypothesis, $\mathscr{A} \models \psi_1$. Hence, also $\mathscr{A} \models \psi_1 \vee \psi_2$. Conversely, assuming $\mathscr{A} \models \psi_1 \vee \psi_2$ we get that $\mathscr{A}$ must satisfy at least one of the ψ_j's, without loss of generality $\mathscr{A} \models \psi_1$, and by inductive hypothesis $\psi_1 \in A_\infty$. Then $\psi_1 \vee \psi_2$ must also appear positively, since otherwise, the subset $\{\psi_1, \neg(\psi_1 \vee \psi_2)\}$ of A_∞ is inconsistent.

$\varphi = \neg\psi$: $\neg\psi \in A_\infty$ iff $\psi \notin A_\infty$ iff (by the inductive hypothesis) $\mathscr{A} \not\models \psi$ iff $\mathscr{A} \models \neg\psi$.

$\varphi = \langle\alpha\rangle\psi$: This case is handled by induction on the structure of α.

$\alpha = x \leftarrow \sigma$: Assume $\langle x \leftarrow \sigma\rangle\psi \in A_\infty$. By the construction of A_∞, it also contains $\psi[\sigma/x]$. By the inductive hypothesis $\mathscr{A} \models \psi[\sigma/x]$, so $\mathscr{A} \models \langle x \leftarrow \sigma\rangle\psi$. The converse is shown similarly.

$\alpha = \psi'?, \beta \cup \gamma, \beta \cap \gamma, \beta;\gamma$: Similar.

$\alpha = \beta^*$: Assume $\langle\beta^*\rangle\psi \in A_\infty$. By the construction of A_∞, also $\psi^i_{\langle\beta\rangle} \in A_\infty$, for some $i \geq 0$. By the inductive hypothesis, $\mathscr{A} \models \psi^i_{\langle\beta\rangle}$; hence, also $\mathscr{A} \models \langle\beta^*\rangle\psi$. Conversely, assume $\mathscr{A} \models \langle\beta^*\rangle\psi$. Then $\mathscr{A} \models \psi^i_{\langle\beta\rangle}$ for some $i \geq 0$. Again this implies that $\psi^i_{\langle\beta\rangle}$ must appear positively in A_∞, and thus $\langle\beta^*\rangle\psi$ must appear positively too, since otherwise $\{\psi^i_{\langle\beta\rangle}, \neg\langle\beta^*\rangle\psi\}$ is an inconsistent subset of A_∞, by the infinitary rule. □

Now the proof of Theorem 3.10 is completed, since $\varphi^+ \in A_\infty$, so $\mathscr{A} \models \varphi^+$; thus, we have exhibited a model for the consistent formula φ^+.

ACKNOWLEDGMENTS. I would like to thank David Harel for many stimulating discussions during which these ideas have developed, as well as for numerous valuable suggestions and remarks which helped to transfer the ideas into (it is hoped) readable form. I would also like to thank Joe Halpern, Moshe Vardi, and an anonymous referee for reading the paper, making valuable suggestions, and pointing out several errors in previous versions.

REFERENCES

(Note: References [33] and [36] are not cited in text.)

1. CHANDRA, A. K. The power of parallelism and nondeterminism in programming. In *Proceedings of IFIP '74*. North-Holland, Amsterdam, Holland, 1974, pp. 461–465.
2. CHANDRA, A. K., KOZEN, D. C., AND STOCKMEYER, L. J. Alternation. *J. ACM 28*, 1 (Jan. 1981), 114–133.
3. CLOCKSIN, W. F., AND MELLISH, C. S. *Programming in Prolog*, Springer-Verlag, New York, 1981.
4. COOK, S. A. Soundness and completeness of an axiom system for program verification. *SIAM J. Comput. 7*, 1 (1978), 70–90.
5. FISCHER, M. J., AND LADNER, R. E. Propositional dynamic logic of regular programs. *J. Comput. Syst. Sci. 18* (1979), 194–211.
6. HAREL, D. First Order Dynamic Logic. In Lecture Notes in Computer Science, vol. 68. Springer-Verlag, New York, 1979.
7. HAREL, D. And/Or programs: A new approach to structured programming. *ACM Trans. Prog. Lang. and Syst. 2*, 1 (Jan. 1980), 1–17.
8. HAREL, D. Dynamic Logic. In *Handbook of Philosophical Logic*, vol. II. Reidel Publishing, Holland, 1984, pp. 497–604.
9. HAREL, D., AND KOZEN, D. C. A programming language for the inductive sets, and applications. *Inf. Control 63* (1985), 118–139.
10. HAREL, D., AND PELEG, D. On static logics, dynamic logics and complexity classes. *Inf. Control 60* (1984), 86–102.
11. HAREL, D., AND PELEG, D. Process logic with regular formulas. *Theoret. Comput. Sci. 38* (1985), 307–322.
12. HAREL, D., AND PELEG, D. More on looping vs. repeating in dynamic logic. *Inf. Proc. Lett. 20* (1985), 87–90.
13. HAREL, D., KOZEN, D. C., AND PARIKH, R. Process logic: Expressiveness, decidability, completeness. *J. Comput. Syst. Sci. 25* (1982), 144–170.
14. HAREL, D., MEYER, A. R., AND PRATT, V. R. Computability and completeness in logics of programs. In *Proceedings of the 9th ACM Symposium on Theory of Computing* (Boulder, Colo., May 2–4). ACM, New York, 1977, pp. 261–268.
15. HITCHCOCK, P., AND PARK, D. M. R. Induction rules and termination proofs. In *Automata, Languages and Programming*, M. Nivat, Ed. North-Holland, Amsterdam, 1973, pp. 253–263.
16. IMMERMAN, N. Relational queries computable in polynomial time. In *Proceedings of the 14th ACM Symposium on Theory of Computing* (San Francisco, Calif., May 5–7). ACM, New York, 1982, pp. 147–152.
17. KEISLER, J. *Model Theory for Infinitary Logic*. North-Holland, Amsterdam, 1971.
18. KOWALSKI, R. *Logic for Problem Solving*. The Computer Science Library, Artificial Intelligence Series, North-Holland, Amsterdam, 1983.
19. KOZEN, D. C. Results on the propositional μ-calculus. In *Proceedings of the 9th ICALP*. Lecture Notes in Computer Science, vol. 140. Springer-Verlag, New York, 1982, pp. 348–359.
20. MANNA, Z. The correctness of nondeterministic programs. *Artif. Intell. 1* (1970), 1–26.
21. MIRKOWSKA, G. On formalized systems of algorithmic logic. *Bull. Acad. Polon. Sci., Ser. Sci. Math. Astron. Phys. 19* (1971), 421–428.
22. MEYER, A. R., AND HALPERN, J. Y. Axiomatic definitions of programming languages: A theoretical assessment. *J. ACM 29*, 2 (Apr. 1982), 555–576.
23. MEYER, A. R., AND PARIKH, R. Definability in dynamic logic. *J. Comput. Syst. Sci., 23* (1981), 279–298.
24. NISHIMURA, H. Descriptively complete process logic. *Acta Inf. 14* (1980), 359–369.
25. NISHIMURA, H. Arithmetical completeness in first-order dynamic logic for concurrent programs. *Publ. RIMS, Kyoto Univ. 17* (1981), 297–309.
26. PARIKH, R. A decidability result for second order process logic. In *Proceedings of the 19th IEEE Symposium on Foundations of Computing Science*. IEEE, New York, 1978, pp. 178–183.

27. PARIKH, R. Propositional logics of programs: New directions. In *Proceedings of the Conference on Foundations of Computer Theory* (Borgholm, Sweden). Lecture Notes in Computer Sciences, vol. 158. Springer-Verlag, New York, 1983, pp. 347–359.
28. PARIKH, R. The logic of games and its applications. *Ann. Disc. Math. 24* (1985), 111–140.
29. PARK, D. M. R. Finiteness is mu-ineffable. *Theor. Comput. Sci. 3* (1976), 173–181.
30. PELEG, D. Communication in concurrent dynamic logic. *J. Comput. Syst. Sci.* (1987), in press.
31. PELEG, D. Concurrent program schemes and their logics, Tech. Rep. CS84-25. The Weizmann Institute of Science, Rehovot, Israel, Nov. 1984.
32. PNUELI, A. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science.* IEEE, New York, 1977, pp. 46–57.
33. PRATT, V. R. Models of program logics. In *Proceedings of the 20th IEEE Symposium on Foundations of Computer Science.* IEEE, New York, 1979, pp. 115–122.
34. PRATT, V. R. A Decidable μ-calculus (Preliminary Report). In *Proceedings of the 22th IEEE Symposium on Foundations of Computer Science.* IEEE, New York, 1981, pp. 421–427.
35. SHAPIRO, E. Y. Alternation and the computational complexity of logic programs. Tech. Rep. CS84-06. The Weizmann Institute of Science, Rehovot, Israel.
36. SHERMAN, R., AND HAREL, D. A combined proof of one-exponential decidability and completeness for PDL, manuscript. The Weizmann Institute of Science, Rehovot, Israel.
37. TARSKI, A. A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Math. 5* (1955), 285–309.
38. TIURYN, J. Unbounded program memory adds to the expressive power of first-order dynamic logic. In *Proceedings of the 22nd IEEE Symposium on Foundation of Computer Science.* IEEE, New York, 1981, pp. 335–339.
39. URZYCZYN, P. "During" cannot be expressed by "after." *J. Comput. Syst. Sci., 32* (1986), 97–104.
40. VARDI, M. Y. The complexity of relational query languages. In *Proceedings of the 14th ACM Symposium on Theory of Computing* (San Francisco, Calif., May 5–7). ACM, New York, 1982, pp. 137–146.
41. VARDI, M. Y., AND WOLPER, P. Yet another process logic. In *Proceedings of the Workshop on Logics of Programs.* Lecture Notes in Computer Science, vol. 164. Springer-Verlag, New York, 1983, pp. 501–512.
42. VARDI, M., AND WOLPER, P. Automata theoretic techniques for modal logics of programs. In *Proceedings of the 16th ACM Symposium on Theory of Computing* (Washington, DC, Apr. 30–May 2). ACM, New York, 1984, pp. 446–456.
43. WOLPER, P., VARDI, M. Y., AND SISTLA, A. P. Reasoning about Infinite Computation Paths. In *Proceedings of the 24th IEEE Symposium on Foundations of Computer Science.* IEEE, New York, 1983, pp. 185–194.

RECEIVED JULY 1984; REVISED OCTOBER 1986; ACCEPTED OCTOBER 1986