並行プログラム系の解析と検証の形式化

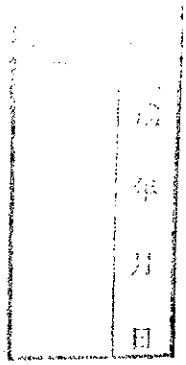
1999年3月

三 [illegible]　哲 也

並行プログラム系の解析と検証の形式化

1999年3月

白銀 哲也



目次

図目次

第 1 章

序論

本論文では並行プログラム系の仕様記述および動作検証を遂行するための形式的体系として tense arithmetic を提案すると共に実時間システムの検証について論ずる。

計算機システムが大規模化・複雑化し、またその需要が増大するに伴って、設計・検証・運用・保守のコストも増大し、信頼性や生産性の低下が懸念されるようになった。「ソフトウェア危機」は叫ばれて久しく、ハードウェア設計開発の現場でも、生産対象の複雑化による生産性の低下は同じく深刻な問題となっている。

これらの対策として様々な手法が模索されており、ソフトウェア生産に関してはオブジェクト指向手法による構成要素のモジュール化や関数型・論理型プログラム言語による比較的高級な表現力の充実が、ハードウェア生産に関しては GUI やエキスパートシステムの導入、シミュレーション技術の向上がある程度の効果をあげてはいるが、生産される計算機システムが大規模・複雑であるために生産分担が細分化される (一人のプログラマだけでアプリケーションの開発を行なえる局面は現在ではごく限られる。) ことから、抜本的な対策には成り得ていない。たとえばソフトウェア部品のモジュール化、ライブラリ化による再利用に関しては蓄積されるのは局所的な技術が主であり、そのままでは他に転用できないことが多い。開発工数は肥大化し、生産現場で必要とされるマン・パワーは増しているのが現状であり、プログラマやハードウェア設計者人口が年々増加する一方であることもそれを裏付けている。

高性能化・高機能化により計算機システムは社会に広く深く浸透するが、要求される性能水準は上がり機能は多様化し、それがハードウェア・ソフトウェアの一層の複雑化・大規模化をうながす。またひとたびシステムに障害・誤動作が発生するとそれが人間生活に与える影響も大きなものとなっている。

こうした背景により、システムの検証や動作解析が必要とされる局面は、

1. 設計段階での検討
2. 開発中の仕様変更
3. 開発後の検証・評価・保守 (障害発生時にはその箇所を理解している人間がいない可能性も高い。)
4. 製品のバージョンアップ

と多く、論理的裏付けのある厳密な手法が必要である。

システムが複雑になれば検証項目も非常に多くなる。正常に運用されているうちは問題無いが予想外の特定の異常な入力があった場合に誤動作を起こしてしまうといった検証漏れはしばし

ば有り得る。検証は効率的に行なわなければならないがマシンアシスタンスの面からみれば、シミュレータを用いた動作テストが可能であっても、複雑なシステムの動作の全ての場合を検証することはシミュレータ自体の制限(シミュレーション時間、分解能)から非実現的であり、テストする項目はある程度絞り込まなければならない。必要なテスト項目の選定に論理的な判断が必要である。更にシミュレータ自体、限られた範囲の問題にしか用いる事が出来ないため、人手に頼った検証を行なわなければならない項目が残ることも多い。

こうした判断は専門家による経験に依るところが多いが、ここで形式的体系を用いることで論理的裏付けが保証される。形式的体系は広汎な問題に適用が可能であることから、

1. 対象のモデル化・抽象化の度合いにより、システムを一部分から全体まで、動作の概略から詳細まで様々なレベルで解析・検証可能になる。論理回路・ソフトウェアから並列計算機システム全体まで対象とできる。

2. ある問題から別の問題へのアナロジーが容易である。まず、ごく大まかに簡略化した問題から検証を始めて全体的な洞察を行ない、必要に応じて徐々に複雑さを増してより正確な検証を完結させる事が出来る、

という2つの利点がある。用いる体系としては並列・分散モデルが平易に扱えて、自動化に適したものが望ましい。

通常は着目する性質(検証すべき定理)に本質的に関係ない動作や属性は切り捨てて対象を抽象化する。イベントが発生するタイミングについての議論は設計の正確さを検証する際に切り離して、独立した問題として扱うのが常であり、そのような体系が多かった。

本論文では形式的体系 tense arithmetic[15] を提示し、並行プログラム表現・解析について述べる。tense arithmetic では、実時間値が全く出現しない抽象的なレベルから実時間値が必要とされる具体的なレベルまで、様々な抽象度での検証が可能となる。特に真価を発揮するのは実際の時間値が提示されている問題についてである。まず実際の時間値の出現しない検証を行ない、時間値の出現する検証に逐次具体化し、複雑かつ意味内容が深い検証を行なうことができる。並行プログラム系の実時間動作の検証は、単にソフトウェアの検証にとどまらず、計算機内外のハードウェアへの応用が可能である。コンピュータネットワーク技術が発達を続ける今日、有限の資源を複数の主体が効率良く利用しなければならない状況は無数に考えられ、またもともとシビアなタイミングが要求される計算機内部(チップ内、あるいはそれらの間や外部に対する効果的なデータ転送)の検証にも有効である。

tense arithmetic は、有理数時間を含む並行プログラム系検証の為に開発された有理数時間を陽に扱う形式的体系であり、命題が(「観察時刻」以後)最初に成立するまでの間隔として解釈する演算“[ ]”と、状態の経過を表す演算子“;”を用いる。形式的体系としては有理数理論および ν- 理論 [8] を論理的基盤としているため公理系が簡潔であり、体系の拡張が容易である。実時間値を証明中に用いる事自体にも証明遂行上の大きな利点がある。また、プロセスケジューラ(実行する処理を示す計算機の制御変数)を一般化した拍車という概念をもとにプログラムを定式化し、推論を行なうことで検証が直観的かつ厳密に遂行できる。

推論にはシーケントを用いた LK[4] と同じ方法を用いる。有理数理論における定理は、tense arithmetic において証明図に始式として取り入れられ、任意の時刻で、任意のプログラムのポストコンディションとして成立するものとして扱われる。有理数理論の定理が導入可能であるために、以下に挙げる利点が生ずる。

1. 有理数理論は強力であり、多くのの時相論理的性質は、公理・推論規則を特別に用意せずとも、有理数理論上の定理に自然に帰着できることが多い。また有理数理論の体系は慣れ親しまれており、扱い易く、その上で推論を遂行することは、論理的に遂行する場合に比べて直観的に理解し易く、推論自体の検証も容易となる。

2. 停止しないプログラムを対象にできる。またTSS的なインターリーブ(interleave、時分割)でなく、真に並行なプロセスを扱える。

これらは、時相論理を基礎とする体系には無い長所である

3. 実際の時間値を証明中に使用可能であり自然数時間体系と異なり量子化に煩わされることがない。

後述のように実時間値を証明中に用いる事自体にも証明遂行上の大きな利点がある。

対象プログラムを定式化して tense arithmetic で扱うために拍車(spur)という概念を導入する。拍車や軌跡については我々のグループの研究全体の基礎となるν-理論でのそれらを概念的基礎としている。拍車は、並行プロセスと1対1対応した、プロセススケジューラを一般化した概念であり、時相論理におけるネクスト演算子"○"を並行プログラム系について厳密化したものと考えることもできる。

検証対象となるプログラムを構成する並行プロセスの各ステップの実行をプログラム公理と呼ぶ論理式として定式化する(抽象化の度合いにより1ステップの規模は様々である)。このとき次の実行ステップへの遷移を制御する拍車を媒介に記述する。この論理式化の手続きは半自動的であり、かつ人間の直観に即したものである。

プログラム検証においては拍車の駆動する系列をグラフオートマトンの入力と見なし、更に各実行ステップに実行時間の上限や下限を割り当て、動作を解析することにより以下の利点が得られる。

1. タイミングチャート、状態遷移図を用いた従来の手法を厳密化した手法であるため、それらの知見を取り入れることができる。

2. 拍車によって計算実行列(computation sequence)をインデックス付け、可能な計算実行列の枝刈りを容易にする。拍車の駆動する系列によって、システム全体の挙動(各構成プロセスのそれと比べて複雑でありしばしば理解が困難)についての場合分けが容易であり、たとえば到達可能な状態と不可能な状態の判別に有効である。

3. 実行時間の幅の伝播の表現・解析が容易である。

実行時間を含む証明では、実行時間に関する制約が増えるため可能な条件分岐が少なくなることで証明が簡単になることが多いことを確認し、純粋に論理的に検証を行なう場合よりも有効であることがわかった。

これらの特徴により tense arithmetic は並行プログラム系の実時間動作の検証について有効な体系となっており、システム開発の現場で行なわれてきたインフォーマルな仕様表現や検証をTAで記述し証明することで厳密化できる。

本論文の構成は以下の通りである。第2章では tense arithmetic の形式的体系を定義する。体系を筆者が示し、共同研究者と共に厳密かつ実際的に構築した。本章の内容は[15]で発表されている。第3章では tense arithmetic を用いた並行プログラム系の定式化から動作解析・定

理証明までの統合的手法 automata spur を紹介し、他の形式的体系との比較も論じる。本章の内容は [25] で発表される。第 4 章では、並行プログラム系検証への適用例を示す。tense arithmetic は OS や分散 DB 等の並行プログラム系だけでなく、複数の主体が並行動作するシステムへ広く適用することが可能である。ここでは Dekker の解と呼ばれる並行プログラム [3, 11]、T 字路における合流問題、自動伴奏システムの検証を挙げる。自動伴奏システムの検証については、CSP[6] と tense arithmetic の関係も述べる。本章の各証明は筆者が行なったものであり、数学的手法での証明は [33, 34, 35] で、tense arithmetic によるものが [36] でそれぞれ発表されている。最後に第 5 章で結論と展望を述べる。

第 2 章

形式的体系 Tense Arithmetic

この章では形式的体系 tense arithmetic (以下 TA と略す) の定義を行なう。

TA は並行プログラム系の検証を行なうための形式的体系である。プログラム変数の値やコントロールの位置を表すプログラムラベルを時刻により真理値の変わる論理式で表現することにより検証を行なう。時刻により変化する論理式を取り扱う上で、2 つの論理式 F_1, F_2 に関して「F_1 の成立する時刻と F_2 の成立する時刻の前後関係」や「F_1 の成立後の F_2 の成立」という概念が重要であると考える。

2.1 有理数体上の Tense

定義 1 (時間) 有理数全体の集合に正の無限遠点 ∞ を添加したものを時間として扱う[1]。ただし ∞ については、

$$r \in \mathcal{Q} \Leftrightarrow r < \infty, r + \infty = \infty + r = \infty \quad every\ r \in \mathcal{Q},$$

$$\inf \emptyset = \infty.$$

を満たすものとする。

TA では、時刻によって値の変化するプログラム変数やそれによって真理値の変化する論理式を扱う。述語論理等の時間の概念をもたない通常の体系では定数の解釈はモデルを固定すると一意であるので、プログラム変数等、時刻によって値の変化する対象はそのままでは取り扱えない。そのため TA では、それらを定数記号として扱い時刻によってモデルを変化させ、新しい定数記号に対する解釈を変化させることにより、値の遷移を表現する。

準備として有理数理論が展開できる適当な理論 $T^{\mathcal{Q}}$ に有限個のプログラム変数 (J, J_1 $\cdots$等) を表すための特別な定数 (*special rational constant*) と、有限個のプログラムラベル (l, l_1, $\cdots$等) や後に述べる拍車 (3.1.1節) を定義する基となる有限個の原子拍車 ($\dot{\alpha}$, $\dot{\beta}$, $\cdots$) を表すための特別な真理値型の定数 (*special boolean constant*) を加え、この様に言語が拡張された理論を $T_C^{\mathcal{Q}}$ とする。

$T_C^{\mathcal{Q}}$ の変数全体の集合を $V^{\mathcal{Q}}$ で表す。さらに、加算無限個の *tense variable* を TA の変数として導入する。tense variable 全体の集合を V^T で表す。

[1]時間の要素が特に時の流れのうちのある一点を示すとき、(通常の意味と同様に) 時刻と呼ぶ。

2.2 構文

定義 2 (tense term) *TA* の項 (*tense term*) を以下のように再帰的に定義する。

1. (*rational tense term*) $T_C^{\mathcal{Q}}$ の項 e は tense term である。特に $T_C^{\mathcal{Q}}$ の変数を *rational variable* と呼ぶ。
2. (*tense variable*) *tense variable* x は tense term である。
3. (*ascent*) P が $T_C^{\mathcal{Q}}$ の論理式のとき $\lfloor P \rfloor$ は tense term である。これを P の立ち上がり (*tense of* P) と呼ぶ。
4. (*futurity*) s が tense term、e が $T_C^{\mathcal{Q}}$ の項、x が tense variable、P が $T_C^{\mathcal{Q}}$ の論理式のとき $s;e$, $s;x$, $s;\lfloor P \rfloor$ はそれぞれ tense term である。";" は経過演算子 (*futurity operator*) と呼ぶ。また tense term $s;s_1$ について s, s_1 をそれぞれ *prefix*, *postfix* と呼ぶ。

定義 3 (tense formula) *TA* の論理式 (*tense formula*) を以下のように再帰的に定義する。

1. P が $T_C^{\mathcal{Q}}$ の論理式のとき、P は tense formula である。
2. s, s_1 がそれぞれ tense term のとき、$s = s_1$, $s \leq s_1$ はそれぞれ tense formula である。
3. x が tense variable、r が rational variable、F,G がそれぞれ tense formula であるとき、$\neg(F)$, $(F) \vee (G)$, $\forall x(F)$, $\forall r(F)$ はそれぞれ tense formula である。

直観的には $\lfloor P \rfloor$ は「(観察時刻以後)P が最初に成立するまでの時間」を表す (2.6節)。よって

- $\lfloor P \rfloor = 0$ は「(観察時刻で)P が成立する。」、
- $\lfloor P \rfloor = 1.5$ は「最初に P が成立するのは 1.5 単位時間後である。」、
- $\lfloor P \rfloor < \infty$ は「P は有限時間内に成立する。」、
- $\lfloor \neg P \rfloor = \infty$ は「P は成立し続ける。」

という意味をそれぞれ持つ。また経過演算子 ";" については、たとえば $\lfloor P \rfloor;\lfloor Q \rfloor$ は命題 P の立ち上がり以後の命題 Q の立ち上がりの時刻を表す。経過演算子の prefix, postfix には有理数を表す定数が現れることもあるが、そのような場合に経過演算は加法に似た性質を持つ[2]。

以下本論文では P, P_1, $\cdots$, Q, Q_1, $\cdots$ は $T_C^{\mathcal{Q}}$ の論理式、r, r_1, $\cdots$ は rational variable、x, y, z は tense variable、s, s_1 は tense term、F, F_1, $\cdots$, G, G_1, $\cdots$ は tense formula をそれぞれ表す記号とする。また、以下の記法を用いる。

1. $\wedge$, $\supset$, $\equiv$, $\exists$ や $<$, $\geq$, $>$ 等の記号を通常の略記法に従い TA に導入する。
2. tense formula $s_1 \leq s_2 \ \wedge \ s_2 \leq s_3$ を

$$s_1 \leq s_2 \leq s_3$$

と略記する。$<$ が入り混じったものについても同様とする。

[2]たとえば TA では tense term $1;2$ と 3 は等しい。

3. ∞ は $\lfloor \mathrm{false} \rfloor$ の略記とする。

4. *coincidental operator*":" を以下の定義により導入する。

$$\begin{aligned} x : P &\overset{\text{def}}{\Leftrightarrow} x = (x; \lfloor P \rfloor), \\ x : (s \leq s_1) &\overset{\text{def}}{\Leftrightarrow} (x; s) \leq (x; s_1), \\ x : (s = s_1) &\overset{\text{def}}{\Leftrightarrow} (x; s) = (x; s_1), \\ x : ((F) \vee (G)) &\overset{\text{def}}{\Leftrightarrow} (x : F) \vee (x : G), \\ x : (\neg(F)) &\overset{\text{def}}{\Leftrightarrow} \neg(x : F), \\ x : (\forall r(F)) &\overset{\text{def}}{\Leftrightarrow} \forall r(x : F), \\ x : (\forall y(F)) &\overset{\text{def}}{\Leftrightarrow} \forall y(x : F). \end{aligned}$$

ただし、二通り以上の解釈ができる場合は上から順に適用するものとする。

$x : F$ は "F holds at x" または単に "F at x" と呼び、 tense formula として「x が表す時刻で F が成立している」と解釈される。 F が成立し始めるのは x より以前でも構わない。$x : F$は古典的プログラム証明論における$x\{F\}$や知識論理における$x * F$と本質的に同値で、 F が x の時点におけるプログラムのポストコンディションであることを意味する。 TA において$x : P$はプログラムの性質を記述する上で多用される表現であるが、ただし物理数学のように時間を直接の媒介変数として表す訳ではなく形式的にはあくまで "$\lfloor\ \rfloor$" と ";" から ":" を定義する立場をとる。逆に ":" を基に TA を再構成して "$\lfloor\ \rfloor$" と ";" を定義された記号として導入することも可能である。

5. 記号の結合の強さを以下のように定義し、曖昧さの無い場合には括弧を適当に省略する。 $+, -$ 等は $T_C^{\mathcal{Q}}$ の演算子である。

(strong) $\quad \{+, -, \textit{etc.}\},\ ;,\ \{=, \leq, <, \textit{etc.}\},\ :,\ \neg,\ \wedge,\ \vee,\ \supset,\ \equiv \quad$ (weak)

2.3 公理系

TA の公理 (*proper axioms*) を以下に示す[3]。

1. *order axioms for* $\leq$:

   (a) *reflection*: $\quad \forall x\ (x \leq x)$,

   (b) *asymmetry*: $\quad \forall xy\ (x \leq y\ \wedge\ y \leq x\ \equiv\ x = y)$,

   (c) *transition*: $\quad \forall xyz\ (x \leq y\ \wedge\ y \leq z \supset x \leq z)$,

2. *axioms for tense*:

   (a) *tense axiom*: $\quad \forall x\ (\exists r\ (x = r)\ \vee\ x = \infty)$,

   (b) *'Never Land' (unreachability) axiom*: $\quad \forall r(r < \infty)$,

   (c) *truth axiom*: $\quad P \equiv \lfloor P \rfloor = 0$,

[3] 2.2節で定義したとおり r, r_1 は、 rational variable であり、 ∞ ($\lfloor \mathrm{false} \rfloor$) の値をとらない。 $=, \leq, <$ は $T_C^{\mathcal{Q}}$ のそれらを TA で ∞ について拡張したものであり、たとえば $0 < 1, \forall r r_1(r + r_1 = r 1_r)$, $J = J_1 \supset J + 1 = J_1 + 1$ が成立する。

(d) *futility axiom*: $\forall x\,(\infty = x;\infty)$,

3. *axioms for futurity* ";":

(a) *present axiom*: $\forall x\,(0 \leq x \supset 0;x = x)$,

(b) *passage axiom*: $\forall xy\,(x \leq x;y)$,

(c) *duration axiom*:
$\forall rr_1\,(0 \leq r_1 \ \wedge\ r;r_1 = r + r_1 \ \vee\ r_1 < 0 \ \wedge\ r;r_1 = r)$,

(d) *prefix substitution axiom*: $\forall xyz\,(x = y \ \supset\ x;z = y;z)$,

4. *axioms for tense symbol* $\lfloor\ \rfloor$:

(a) *idempotent axiom*: $\lfloor P \rfloor = \lfloor P \rfloor;\lfloor P \rfloor$,

(b) *precedence axiom*: $\forall x\,(0 \leq x \ \wedge\ x : P \ \supset\ \lfloor P \rfloor \leq x)$,

(c) *advance consequence axiom*: $P \supset Q$ が $T_C^{\mathcal{Q}}$ の定理ならば、$\lfloor Q \rfloor \leq \lfloor P \rfloor$ は TA の公理である。

(d) *monotonicity axiom*: $\forall xy\,(x \leq y \ \supset\ x;\lfloor P \rfloor \leq y;\lfloor P \rfloor)$,

5. *theorems of* $T_C^{\mathcal{Q}}$ *as axioms*: $T_C^{\mathcal{Q}}$ の定理は TA の公理である。

TA では経過演算子 ";" の postfix において代入の公理は成立しない。観察時刻 t で $x = y$ が成立する場合を考えると、t で観察した z が表す時刻で $x = y$ が成立するとは限らない。すなわち観察時刻 t で $x = y$ が成立しても、公理 (3d) とは対照的に、t で観察した $z;x = z;y$ が成立するとは限らない。

TA では検証対象のプログラムを定式化し、公理と同様に証明図の始式として用いる (定義 7)。そのため定式化して得られた tense formula をプログラム公理と呼ぶ。

定義 4 (プログラム公理 (**program axiom**)) 任意の closed tense formula をプログラム公理と呼ぶ。

2.4 推論規則

推論にはシーケントを用いた LK[4] と同じ方法を用いる。

定義 5 (シーケント (**sequent**))

$$F_1,\ \cdots,\ F_m \to G_1,\ \cdots,\ G_n$$

は (TA の) シーケントである。ただし $m+n \geq 0$ とする。$m = 0$ の場合は矢印の左辺が 'true'、$n = 0$ の場合は矢印の右辺が 'false' とみなす。

TA の推論規則は LK の構造, $\neg$, $\forall$ に関する規則と同じもの (付録参照。$\forall$ については rational variable と tense variable の各々に対しての規則) および、次に示す *the homogeneous inference rule* である[4]。

[4] $\wedge, \supset$ の規則は他の LK の推論規則から導ける。

定義 6 (the homogeneous inference rule)

$$\frac{\rightarrow\ F\ \wedge\ 0 \leq s_1\ \wedge \cdots \wedge\ 0 \leq s_n}{\rightarrow\ x : F}$$

ただし $s_1, \cdots, s_n$ は F に現われる等式・不等式中の tense term の第 1 項 (最左の経過演算子 ";" の prefix) とする。

この規則がプログラム検証の最中で用いられる場合、上式の制限 $(0 \leq s_1\ \wedge \cdots \wedge\ 0 \leq s_n)$ は実際には考慮する必要がない。プログラムが定式化される際に、個々の処理単位について実行時間が負であるとして定式化されることが無いためである。

TA の導出 (*derivation*) は、シーケントで構成された木構造とする。

定義 7 (始式 (initial sequent)) *TA* の導出における始式 (木構造の葉) は、以下のいずれかとする。

1. *identic sequent*: $F \rightarrow F$ は始式である。
2. *homogeneous sequent*: $P_1\ \wedge\ \cdots\ \wedge\ P_m\ \supset\ Q_1\ \vee\ \cdots\ \vee\ Q_n$ が T_C^Q の定理ならば、

$$x : P_1,\ \cdots,\ x : P_m \rightarrow x : Q_1,\ \cdots,\ x : Q_n$$

は始式である。特に、$P_1 \wedge \cdots \wedge P_m \supset \mathrm{false}$ (すなわち $\neg(P_1 \wedge \cdots \wedge P_m)$) が T_C^Q の定理ならば、

$$x : P_1,\ \cdots,\ x : P_m\ \rightarrow\ x : \mathrm{false}$$

は始式である。これにより T_C^Q で導出可能な定理は、TA において任意のプログラムのポストコンディションとして成立する。

3. *axiomatic sequent*: A が TA の公理またはプログラム公理 (2.3節) ならば $\rightarrow\ A$ は始式である。

定義 8 (証明図) 次の条件を満たすような有限個のシーケントからつくられる図形を *TA* の証明図という。

1. この証明図のどんなシーケントも、終式とよばれる木構造の葉以外は *TA* の推論規則の上式となっている。
2. すべてのシーケントとはたかだか一つの *TA* の推論規則の上式になっている。
3. 一つのシーケントからその下式へと進んでいってもとのシーケントに戻ることは無い。

TA の導出において終式 $\mathcal{S}$ は導出可能 (*derivable*) であるという。
特に

$$\rightarrow F$$

の形のシーケントが導出可能であるとき、F を証明可能、または TA の定理であるという。証明中にプログラム公理 $A_1, \ldots, A_n$ が始式として用いられたならば

$$A_1, \ldots, A_n \vdash F$$

で表す。プログラム公理が始式として用いられずに導出された場合、$\vdash F$ で表す。

2.5 定理

TA の基本的な定理を示す [15]。これらの定理群は、検証するプログラムに依存しない。

定理 1 *the equality theorems*: $=$ は経過演算子 ";" の postfix の代入則以外について通常の等号の公理を満たす。

定理 2 *the liner order theorems*: TA の不等号 $\leq$ は tense 上の全順序であり、$T_C^{\mathcal{Q}}$ の不等号 $\leq$ を ∞ について自然に拡張したものである。
(a) $\forall xy\ (x \leq y \vee y \leq x)$, (b) $\forall x\ (x \leq \infty)$, (c) $\forall r\ (r < \infty)$.

(証明)
(c), (b), (a) の順に証明する。

(c) Never Land ax.(2b) である。

(b) the tense ax.(2a) と上記 (c) より導かれる。

(a) the tense ax.(1a) には $x = \infty$ の場合と $x = r$ (r は有理数) の場合がある。$x = \infty$ または $y = \infty$ の場合は上に示した (b) より証明される。$x = r$, $y = r_1$ (r, r_1 は有理数) の場合は $r \leq r_1 \ \vee \ r_1 \leq r$ が $T_C^{\mathcal{Q}}$ の定理であり、the equality theor. (1) を用いる。

(証明終)

定理 3 *the prefix substitution theorems*:
(a) $\forall xyz\ (x = y \ \supset \ x;z = y;z)$,
(b) $\forall xy\ (x = y \ \wedge \ x : P \ \supset \ y : P)$.

(証明)
(a) は the prefix substitution ax.(3d) より直接導かれる。

(b) 上記 (a) の z として $\lfloor P \rfloor$ を代入すると

$$x = y \ \supset \ x;\lfloor P \rfloor = y;\lfloor P \rfloor$$

よって the equality theor.(1) より

$$x = y \ \wedge \ x = x;\lfloor P \rfloor \ \supset \ y = y;\lfloor P \rfloor$$

(証明終)

以後、証明中で特に強調する必要の無い場合には、the equality theorems の使用について省略する。

定理 4 *the ascendant theorems*:
(a) $\forall x\ (\lfloor P \rfloor = x \ \supset \ x : P)$, (b) $\forall x\ (x \leq \lfloor P \rfloor \ \supset \ x;\lfloor P \rfloor \leq \lfloor P \rfloor)$,
(c) $0 \leq \lfloor P \rfloor$, (d) $\forall x\ (0 \leq x \ \supset \ \lfloor P \rfloor \leq x;\lfloor P \rfloor)$.

(証明)

(a) the prefix substitution theor.(3b) より

$$\lfloor P \rfloor = x \ \wedge \ \lfloor P \rfloor : P \ \supset \ x : P$$

が導かれる。ここで the idempotent ax.(4a) と coincidental operator":" の定義から $\lfloor P \rfloor : P$ が導かれる。

(b) the monotonicity ax.(4d) より

$$x \leq \lfloor P \rfloor \ \supset \ x;\lfloor P \rfloor \leq \lfloor P \rfloor;\lfloor P \rfloor$$

が導かれる。ここで the idempotent ax.(4a) から $\lfloor P \rfloor;\lfloor P \rfloor = \lfloor P \rfloor$ が導かれる。

(c) $P \supset \mathrm{true}$ が $T_C^{\mathcal{Q}}$ の定理であるため、the advance consequence ax.(4c) より

$$\lfloor \mathrm{true} \rfloor \leq \lfloor P \rfloor$$

が導かれる。ここで the truth ax.(2c) より $\lfloor \mathrm{true} \rfloor = 0$ が導かれる。

(d) the monotonicity ax.(4d) より

$$0 \leq x \ \supset \ 0;\lfloor P \rfloor \leq x;\lfloor P \rfloor$$

が導かれる。上記 (c) により $0 \leq \lfloor P \rfloor$ であるため the present ax.3(a) から $0;\lfloor P \rfloor = 0$ が導かれる。

(証明終)

定理 5 *the monotonisity theorems*:
経過演算子 ";" には限定的にではあるが単調性がある。
(a) $\forall xy\ (x \leq x;y)$, (b) $\forall xy\ (x;y \leq y \ \supset \ x \leq y)$, (c) $\forall xy\ (y = x;y \ \supset \ x \leq y)$.

(証明)

(a) これは the passage ax.(3b) である。

(b) 推移律により

$$x \leq x;y \ \wedge \ x;y \leq y \ \supset \ x \leq y$$

であるが、the passage ax.(3b) より $x \leq x;y$ が導かれる。

(c) the equality theor.(1) より、$y = x;y \ \supset x;y \leq y$ が導かれる。ここで $y = x;y$ のとき $x \leq x;y$ が導かれるため、上記 (b) より $x \leq y$ が成立する。

(証明終)

定理 6 *the theorems on tense constants*:
(a) $\infty : P$, (b) $\forall x\ (x;0 = x)$, (c) $\forall x\ (\infty;x = \infty)$.

(証明)
(c), (a), (b) の順に証明する。

(c) the passage ax.(3b) より $\infty \leq \infty; x$ が導かれる。また the linear order theor.(2b) より $\infty; x \leq \infty$ が導かれる。よって $\infty = \infty; x$ が成立する。

(a) 上記 (c) の x として $\lfloor P \rfloor$ を代入すると、$\infty = \infty; \lfloor P \rfloor$ が得られる。これは定義より $\infty : P$ (「時刻 ∞ で P が成立する」) である。

(b) $x = \infty$ の場合は、上記 (c) よりただちに $\infty; 0 = \infty$ が導かれる。$x = r$ の場合は、the duration ax.(3c) より $r; 0 = r$ であり、the prefix substitution ax.(3d) を用いて $x; 0 = r; 0$ が導かれる。

(証明終)

定理 7 *the rejection theorems*:
(a) $\forall x\ (x < \infty \supset \neg x : P \vee \neg x : \neg P)$,
(b) $\forall x\ (x : P \vee x : \neg P)$,
(c) $\forall x\ (x < \infty \supset x; \lfloor P \rfloor \neq x; \lfloor \neg P \rfloor)$,
(d) $\lfloor P \rfloor \neq \lfloor \neg P \rfloor$,
(e) 'present negative': $\forall x\ (0 \leq x < \lfloor P \rfloor \supset x : \neg P)$,
(f) 'future negative': $\forall x\ (x < x; \lfloor P \rfloor \supset x : \neg P)$
(g) $\forall x\ (x < \infty \supset \neg x : \text{false})$.

(証明)
(g), (a), (b), (c), (d), (f), (e) の順に証明する。

(g) the futility ax.(2d) により前提は $x < x; \infty$ となる。定義により $\neg x : \text{false}$ が得られる[5]。

(a)

$$\dfrac{\dfrac{}{\to x < \infty \supset \neg x : \text{false}}\ \text{(rejection theor.(7g))} \qquad \dfrac{P, \neg P \to}{x : P, x : \neg P \to x : \text{false}}\ \text{(homogeneous)}}{x < \infty \to \neg(x : P), \neg(x : \neg P)}$$

(補足)
$T_C^{\mathcal{Q}}$ の定理 $\neg(P \wedge \neg P)$ により、証明図 2 行目の the initial homogeneous sequent

$$x : P, x : \neg P \to x : \text{false}$$

が得られる。これがこの証明の始式の一つとなっているが、可読性のため、この始式の拠り所となった $\neg(P \wedge \neg P)$ をシーケント

$$P, \neg P \to$$

の形であたかもそれが始式であるかのように、追加して記述した。なお証明図中の二重線は、LK の構造に関する規則 (cut を含む) および the equality theorems のみを複数回用いたことを示す。

[5] $\neg x : \text{false}$ は括弧を補えば $\neg(x : \text{false})$、すなわち $x \neq x; \text{false}$ である。

(b) 上記 (a) の証明と同様に、T_C^Q の定理として $\neg(P \wedge \neg P)$ のかわりに $P \vee \neg P$ を用いて導く。

(c) 上記 (a) (b) より、

$$x < \infty \supset x : P \wedge \neg x : \neg P \vee x : \neg P \wedge \neg x : P$$

が導かれる。よって

$$x < \infty \supset x = x; \lfloor P \rfloor \wedge x \neq x; \lfloor \neg P \rfloor \vee x = x; \lfloor \neg P \rfloor \wedge x \neq x; \lfloor P \rfloor$$

が得られ、

$$x < \infty \supset x; \lfloor P \rfloor \neq x; \lfloor \neg P \rfloor$$

が導かれる。

(d) 上記 (c) により、

$$0; \lfloor P \rfloor \neq 0; \lfloor \neg P \rfloor$$

である。一方、the ascendant theor.(4c) より $0 \leq \lfloor P \rfloor$ が得られ、the present ax.(3a) より

$$0; \lfloor P \rfloor = \lfloor P \rfloor$$

が導かれる。

(f) 上記 (b) により、

$$\neg x : \lfloor P \rfloor \supset x : \neg P$$

が導かれる。coincidental operator":"の定義により $\neg x : \lfloor P \rfloor \equiv x < x; \lfloor P \rfloor$ である。

(e) the ascendant theor.(4d) により、

$$0 \leq x \supset \lfloor P \rfloor \leq x; \lfloor P \rfloor$$

が導かれる。前提は、推移律により $x \leq x; \lfloor P \rfloor$ となり、上記 (f) から $x : P$ が導かれる。

(証明終)

直観的に the rejection theor. (7a)(7b) はそれぞれ「任意の観察時刻で P または $\neg P$ が成立する。」「有限の時刻では false は成立しない。」という事実を意味する。(7a)(7b)(7g) により、有限時間における一種の無矛盾性が示せた。なお無限遠点 ∞ においては $P \wedge \neg P$ が成立する (the theorems tense constants (6a))。

定理 8 *the coincidence theorems*:
次の 3 つの論理式は互いに同値である。
(i) $\lfloor P \rfloor : Q$, (ii) $\lfloor P \rfloor : P \wedge Q$, (iii) $\lfloor P \rfloor = \lfloor P \wedge Q \rfloor$.

(証明)

(i) ⇒ (ii).

the idempotent ax.(4a) により $\lfloor P \rfloor : P$ が導かれる。homogeneous sequent として

$$\lfloor P \rfloor : P \ \wedge \ \lfloor P \rfloor : Q \ \supset \quad \lfloor P \rfloor : P \wedge Q$$

が得られる。

(ii) ⇒ (iii).

the precedence ax.(4b) と the ascendant theor.(2.4c) より

$$\lfloor P \rfloor : P \wedge Q \ \supset \lfloor P \wedge Q \rfloor \leq \lfloor P \rfloor$$

が導かれる。$P \wedge Q \supset P$ が $T_C^{\mathcal{Q}}$ の定理であるため the advance consequence ax.(4c) より

$$\lfloor P \rfloor \leq \lfloor P \wedge Q \rfloor$$

が導かれる。よって、$\lfloor P \rfloor = \lfloor P \wedge Q \rfloor$ が得られる。

(iii) ⇒ (i).

$P \wedge Q \supset Q$ が $T_C^{\mathcal{Q}}$ の定理であるので homogeneous sequent として

$$\lfloor P \wedge Q \rfloor : P \wedge Q \ \supset \ \lfloor P \wedge Q \rfloor : Q$$

が得られる。the idempotent ax.(4a) より $\lfloor P \wedge Q \rfloor : P \wedge Q$ と $\lfloor P \wedge Q \rfloor : Q$ が得られる。よって、仮定と the prefix substitution theor.(3b) より (i) が成立する。

(証明終)

2.6 意味論と健全性

$\mathcal{Q}$ を有理数のスタンダードモデルとする。$T^{\mathcal{Q}}$ の代入 $\rho^{\mathcal{Q}}$ は、$\rho^{\mathcal{Q}} : V^{\mathcal{Q}} \rightarrow \mathcal{Q}$ である[6]。$T^{\mathcal{Q}}$ のモデル $\mathcal{Q}$ に、special rational constant と special boolean constant の解釈を加えて拡大したものを $T_C^{\mathcal{Q}}$ のモデル $\mathcal{M}$ とする。各々の special constant の解釈は、異なったモデルでは異なった値を取り得る。

ある観察時刻で論理式を解釈するには、その論理式が時間概念を扱う述語を含む場合、その時刻のモデルだけでなくそれ以降すべての時刻のモデルが必要である。時間によるプログラムの状態変化を形式的体系に取り入れるために、すなわち時刻によりモデルを変えて解釈するため軌跡 (locus) と呼ぶ、時刻 t から $T_C^{\mathcal{Q}}$ のモデル $\mathcal{M}$ への関数 χ を用いる。

ここで $\mathcal{E}$ を前述のモデル $\mathcal{M}$ の集合とすると、$\chi : \mathcal{Q} \rightarrow \mathcal{E}$ で表される。この軌跡は、ν- 理論 [8, 38] の軌跡の概念を基礎としている。軌跡は計算の経過 (computation sequence) を一般化したものでありプログラムの実行によるプログラム変数やコントロールの変遷を表す[7]。また、時刻 t での special rational constant J、special boolean constant l, $\dot{\alpha}$ の値を各々 $\chi(t,\ J), \chi(t,\ l), \chi(t,\ \dot{\alpha})$ で表す。

---

[6] 2.1節で示したように $T_C^{\mathcal{Q}}$ の変数全体の集合を $V^{\mathcal{Q}}$ で、tense variable 全体の集合を V^T でそれぞれ表す。

[7] ν- 理論の「行動は無限小時間内に行われる」という無限小性の公理 [16] よりプログラム変数の値とプログラムラベルや原子拍車の真理値の変化は瞬時に行われるものとする。

TA における代入 ρ は、関数の対 $\langle \rho^{\mathcal{Q}}, \rho^T \rangle$ とし、$\rho^{\mathcal{Q}}$ は $T^{\mathcal{Q}}$ における代入、ρ^T は $\rho^T : \mathcal{Q} \to V^T \to \mathcal{Q}+\{\infty\}$ のタイプの関数とする。すなわち、tense variable x の時刻 t の値は $\rho^T(t,\ x)$ と表される。また、$\rho^{\mathcal{Q}}(e)$、$\rho^{\mathcal{Q}}(P)$ は $\rho^{\mathcal{Q}}$ での変数の値を代入して得られる式および論理式の値を表す。

$T^{\mathcal{Q}}$ における代入 $\rho^{\mathcal{Q}}$ と軌跡 χ と時刻 $t(\in \mathcal{Q})$ が与えられれば、$T^{\mathcal{Q}}_C$ の論理式 P の値が決定される。P が $\rho^{\mathcal{Q}}$、χ、t のもとで成り立つ、すなわち $\chi(t,\ \rho^{\mathcal{Q}}(P)) = \text{true}$ であることを、

$$\rho^{\mathcal{Q}}, \chi(t) \models P$$

で表す。

プログラムの動作は（連続的に変化する外界の対象について制御する場合にも）離散的であるため離散性の公準 (*discreteness postulate*) をおく。

定義 9 (離散性 (**discreteness**)) 有理数の上昇列 $(r_i)_{i=0,1,2,\cdots}$ が有限であるか、任意に大きな有理数を含むとき、**離散的** (*discrete*) であるという。

定義 10 (離散的階段関数 (**discrete step function**)) 時刻から任意の集合 S への関数 $f : \mathcal{Q} \to S$ が、ある離散的な有理数列 $(r_i)_{i=0,1,2,\cdots}$ と集合 S の要素 x_i に対して $f(t) = x_i$ （ただし $r_i < t \leq r_{i+1}$）を満たすとき、**離散的階段関数** (*discrete step function*) という。

離散性の公準 (discreteness postulate)。全ての軌跡 χ と全ての V^T への代入 ρ^T は離散的階段関数になるものとする。

離散性の公準により、対象プログラムの状態が離散的に変化することと変化する点が有理数となることが保証される。

TA の代入 ρ、軌跡 χ、観察時刻 $t(\in \mathcal{Q})$ を定めると tense term s は $\mathcal{Q}+\{\infty\}$ の値として以下のように解釈される。

定義 11 (**tense term の解釈**) 与えられた ρ, χ、t に対して、tense term の解釈 $\sim$ は、その構造（定義 2）に基づき以下のように帰納的に定義される。

1. $e^{\sim}_{\chi,\rho,t} \stackrel{\text{def}}{=} r, \quad \textit{such that } t = \inf\{u \mid t < u,\ \rho^{\mathcal{Q}}, \chi(u) \models r = e\},$
2. $x^{\sim}_{\chi,\rho,t} \stackrel{\text{def}}{=} w, \quad \textit{such that } t = \inf\{u \mid t < u,\ w = \rho^T(u,\ x)\}, \quad (\inf \emptyset = \infty),$
3. $$\lfloor P \rfloor^{\sim}_{\chi,\rho,t} \stackrel{\text{def}}{=} \begin{cases} \inf\{u \mid t < u,\ \rho^{\mathcal{Q}}, \chi(u) \models P\} - t & \textit{if } \{u \mid t < u,\ \rho^{\mathcal{Q}}, \chi(u) \models P\} \neq \emptyset, \\ \infty & \textit{otherwise,} \end{cases}$$
4. $$(s; s_1)^{\sim}_{\chi,\rho,t} \stackrel{\text{def}}{=} \begin{cases} s^{\sim}_{\chi,\rho,t} + s_1{}^{\sim}_{\chi,\rho,t+s^{\sim}_{\chi,\rho,t}} & \textit{if } s^{\sim}_{\chi,\rho,t} < \infty \\ & \quad \textit{and } 0 \leq s_1{}^{\sim}_{\chi,\rho,t+s^{\sim}_{\chi,\rho,t}} < \infty, \\ s^{\sim}_{\chi,\rho,t} & \textit{if } s^{\sim}_{\chi,\rho,t} < \infty \\ & \quad \textit{and } s_1{}^{\sim}_{\chi,\rho,t+s^{\sim}_{\chi,\rho,t}} < 0, \\ \infty & \textit{otherwise.} \end{cases}$$

ここで、s_1 は e, x, $\lfloor P \rfloor$ のいずれかである。

$T_C^{\mathcal{Q}}$ の項 e、tense variable x、$T_C^{\mathcal{Q}}$ の論理式 P の立ち上りはそれぞれ、観察時刻 t の正の近傍での値として解釈される。離散性の公準により inf によって得られた値が $\mathcal{Q}\cup\{\infty\}$ の要素として存在することが保証されている。

定義 12 (tense formula の解釈) 与えられた ρ, χ、t に対して、tense formula の解釈 $\#$ は、その構造 (定義 3) に基づき以下のように帰納的に定義される。

$$
\begin{array}{lrcl}
1. & P^{\#}_{\chi,\rho,t} = \text{true} & \stackrel{\text{def}}{\Leftrightarrow} & \lfloor P\rfloor^{\sim}_{\chi,\rho,t} = 0, \\
2. & (s = s_1)^{\#}_{\chi,\rho,t} = \text{true} & \stackrel{\text{def}}{\Leftrightarrow} & s^{\sim}_{\chi,\rho,t} = s_1{}^{\sim}_{\chi,\rho,t}, \\
 & (s \le s_1)^{\#}_{\chi,\rho,t} = \text{true} & \stackrel{\text{def}}{\Leftrightarrow} & s^{\sim}_{\chi,\rho,t} \le s_1{}^{\sim}_{\chi,\rho,t}, \\
3. & (\neg F)^{\#}_{\chi,\rho,t} = \text{true} & \stackrel{\text{def}}{\Leftrightarrow} & F^{\#}_{\chi,\rho,t} = \text{false}, \\
4. & (F\vee G)^{\#}_{\chi,\rho,t} = \text{true} & \stackrel{\text{def}}{\Leftrightarrow} & F^{\#}_{\chi,\rho,t} = \text{true } \mathit{or}\ G^{\#}_{\chi,\rho,t} = \text{true}, \\
5. & (\forall x F)^{\#}_{\chi,\rho,t} = \text{true} & \stackrel{\text{def}}{\Leftrightarrow} & F^{\#}_{\chi,(\rho^{\mathcal{Q}},\ \rho'^{T}),t} = \text{true}, \quad \mathit{every}\ \rho'^{T} \\
 & & & \mathit{such\ that}\ \rho'^{T}(u,\ y) = \rho^{T}(u,\ y) \\
 & & & \mathit{each}\ u\ \mathit{and\ each}\ y \in V^{T}\ \ \mathit{other\ than}\ x, \\
6. & (\forall r F)^{\#}_{\chi,\rho,t} = \text{true} & \stackrel{\text{def}}{\Leftrightarrow} & F^{\#}_{\chi,(\rho'^{\mathcal{Q}},\ \rho^{T}),t} = \text{true}, \quad \mathit{every}\ \rho'^{\mathcal{Q}} \\
 & & & \mathit{such\ that}\ \rho'^{\mathcal{Q}}(v) = \rho^{\mathcal{Q}}(v) \\
 & & & \mathit{each}\ v \in V^{\mathcal{Q}}\ \ \mathit{other\ than}\ r.
\end{array}
$$

定義 13 以下の式が成立する時、軌跡 χ が F を充足する (χ *satisfies* F) という。

$$F^{\#}_{\rho,\chi}(t) = \text{true}, \qquad \mathit{every}\ \rho\ \ \mathit{and}\ t,$$

このとき、$\chi \models F$ と表す。

定義 14 シーケント $\mathcal{S}$ が

$$F_1, \ldots, F_m \ \rightarrow\ G_1, \ldots, G_n$$

であるとき、軌跡 χ が

$$\chi \models \neg F_1 \vee \cdots \vee \neg F_m \vee G_1 \cdots \vee G_n$$

を満たすならばそのときに限り χ は $\mathcal{S}$ を充足する (χ *satisfies* $\mathcal{S}$) といい、$\chi \models \mathcal{S}$ で表す。

検証対象のプログラム $\mathbf{P}$ を定式化したプログラム公理全体の集合を $\Gamma_{\mathbf{P}}$ で表す。$\Gamma_{\mathbf{P}}$ に含まれるプログラム公理をすべて充足する軌跡の集合を $X_{\mathbf{P}}$ で表す。

$$X_{\mathbf{P}} \stackrel{\text{def}}{=} \{\chi \mid \chi \models A,\ A \in \Gamma_{\mathbf{P}}\}$$

$A_1, \ldots, A_n \in \Gamma_{\mathbf{P}}$ とし、それらから tense formula F の導出が存在するならば ($A_1 \wedge \ldots \wedge A_n \vdash F$)、$\Gamma_{\mathbf{P}}$ に属する全てのプログラム公理を充足する軌跡は明らかに F を充足する。すなわちプログラム $\mathbf{P}$ のプログラム公理 $A_1, \ldots, A_n$ から証明された F は、$\mathbf{P}$ の性質として成り立つ。

定義 15 シーケント $\mathcal{S}_1, \ldots, \mathcal{S}_m$ を充足する任意の軌跡 χ がシーケント $\mathcal{S}$ を充足するときそのときに限り、$\mathcal{S}_1, \ldots, \mathcal{S}_m$ から $\mathcal{S}$ の導出が健全 (*sound*) であるという。

TA の各推論規則は健全である。LK の規則を用いたものについては自明であり、the homogeneous inference rule は、上式で任意の観察時刻について F が真であると仮定する事になるので、下式で F の観察時刻を任意の x に変更して解釈する $x : F$ は成立する。また TA の proper axiom は、任意の軌跡 χ によって充足される。よって、F が証明可能なとき、任意の軌跡 χ が F を充足することが保証される。すなわち TA は健全であり、無矛盾である。

第 3 章

拍車を用いた検証

プログラム検証を行なうために、検証対象となるプログラムを、それを構成する並行プロセスの各ステップの仕様についてプログラム公理と呼ぶ論理式として定式化する。

3.1 並行プログラムの定式化

3.1.1 拍車

対象プログラムを定式化して TA で扱うために拍車 (spur) という概念を導入する。拍車は軌跡と同様に ν- 理論 [8, 38] での拍車を概念的基礎とする。

拍車は、$T_C^{\mathcal{Q}}$ の原子拍車と呼ぶ special boolean constant $\dot{\gamma}$ に対して $\lfloor\dot{\gamma}\rfloor;\lfloor\neg\dot{\gamma}\rfloor$ として定義される tense term であり γ で表される[1]。この定義は、実時間値が陽に与えられなくとも推論を可能とするために採用されている。拍車 γ が2 回駆動してプロセスが 2 ステップ進む時間は $\gamma;\gamma$ で表されるが、γ の定義と軌跡の離散性から、任意の時刻で $0 < \gamma$ かつ ($\lfloor\dot{\gamma}\rfloor = \lfloor\dot{\gamma}\rfloor;\lfloor\dot{\gamma}\rfloor$ が成立するのに対し) $\gamma < \gamma;\gamma$ が保証される。

以後 tense formula 中で拍車と拍車の間の経過演算子 ";" は省略して表記する。

プログラム公理中では原子拍車 $\dot{\gamma}$ ではなく拍車 γ の時間的な性質が示される。たとえばプログラム公理

$$l_0 \supset \alpha \leq 0.9 \wedge \alpha = \lfloor l \rfloor$$

は「ラベル l_0 で表される実行ステップにコントロールがあるとき、拍車 α が 0.9 単位時間以内に駆動し、それによってコントロールはラベル l で表される実行ステップに移る。」という性質を論理式化したものであるが、l_0 のとき $\alpha \leq 0.9$ すなわち $\lfloor\dot{\alpha}\rfloor;\lfloor\neg\dot{\alpha}\rfloor \leq 0.9$ によって

$$\lfloor\dot{\alpha}\rfloor < 0.9 \ \wedge \ \lfloor\neg\dot{\alpha}\rfloor \leq 0.9$$

が導かれる。このように、次に拍車 γ の駆動するまでの時間の上限が示された場合、原子拍車 $\dot{\gamma}$ がそれまでに立ち上がり、その後立ち下がることを意味する。

拍車は、並行プロセスと 1 対 1 対応した、プロセススケジューラを一般化した概念であり、時相論理におけるネクスト演算子 "○" を並行プログラム系について細分化・厳密化したものと考えることもできる。

TA では、検証の対象となるプログラムの仕様をプログラム公理と呼ぶ論理式で表す。また、プログラム公理を一つ矢印の右辺にもつシーケントを始式として用いることにより証明を行な

[1]本論文では $\gamma, \dot{\gamma}$ はそれぞれ拍車、原子拍車を特に一般的に示す。

う。プログラム公理は, プログラム変数、その値、現在のプログラムラベル、次のプログラムラベル、拍車等を含む。

プロセスと拍車は1対1で対応づけられ、あるプロセスにおけるプログラム変数の値の変化は、そのプロセスに対応する拍車が真から偽に変化する時刻に起こるものとする (無限小性の公理により値の変化は一瞬で行われる)。プロセスは対応する拍車で駆動するととらえることもできる。また、プロセスは、それに対応付けられた拍車を用いて α 駆動のプロセス (α-driven process) または α 系のプロセス、β 駆動のプロセス (β-driven process) または β 系のプロセス等と表現されることがある。

ここで、拍車 γ の例を示す (図 3.1)。

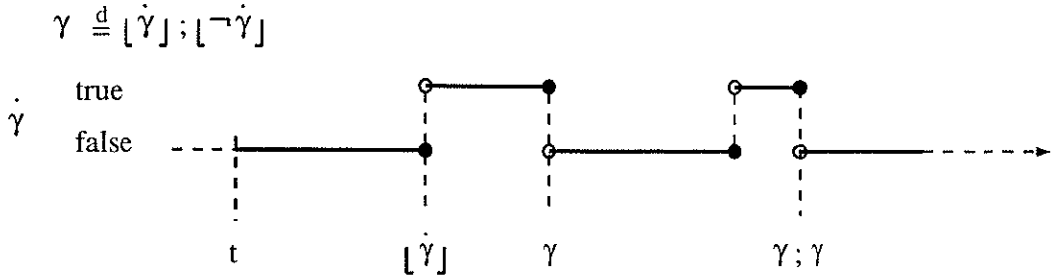


図 3.1: 拍車の例

3.1.2 プログラム公理

各プロセスには1対1対応で拍車が対応づけられる。さらに、各プロセスを処理ブロック (実行ステップとも言う) に分け、それぞれのブロックにラベルをふり、ブロック間の遷移やプログラム変数に対する代入、他のプロセスとの通信等を公理化する。こうしたブロックはしばしばそれに対応づけられたラベルで呼ばれる。

TA によるプログラム検証においては、検証対象のプログラム系を構成する並行プロセスの各処理ブロックの実行をプログラム公理と呼ぶ論理式として定式化する (抽象化の度合いにより1ブロックの規模は様々である)。このとき次に実行される処理ブロックへの遷移を制御する拍車を媒介に記述する。処理ブロックの機能と定式化について代表的なものを以下に挙げる。

1. (ラベルの性質)
   一つのプロセスに属するプログラムラベルの値は、常にただ一つだけが真となる。

2. (実行時間の上限と下限) γ 駆動のプロセス内の l_0 の実行時間の上限・下限がそれぞれ K_{max}, K_{min} であるならばプログラム公理では

   $$l_0 \supset \gamma \leq K_{max},$$
   $$\neg l_0 \supset l_0 : K_{min} \leq \gamma$$

   と表される。

3. (単純な遷移)
   図 3.2に示すように α 駆動のプロセス内の l_0 のステップの次に (分岐がなく) l_1 のステップが実行されることは次のように定式化される。

   $$l_0 \supset \alpha = \lfloor l_1 \rfloor, l_1 \supset \alpha = \lfloor l_2 \rfloor, \cdots$$

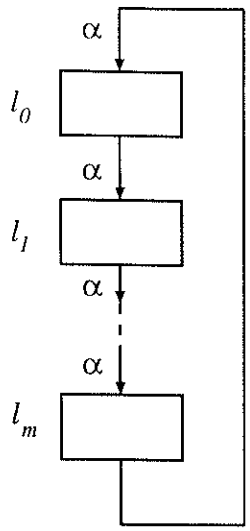


図 3.2: 単純な遷移

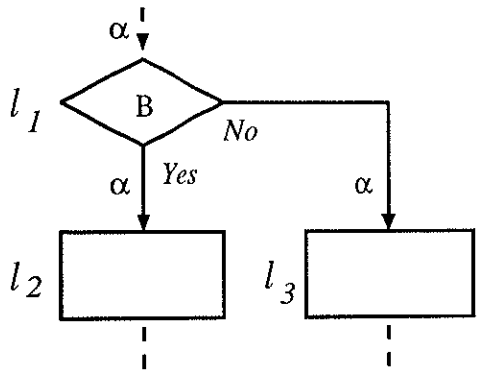


図 3.3: 条件分岐

4. (条件分岐) 図 3.3に示す、条件 B による単純な分岐は次のようにプログラム公理化できる。

$$l_1 \wedge \alpha : B \supset \alpha = \lfloor l_2 \rfloor,$$
$$l_1 \wedge \alpha : \neg B \supset \alpha = \lfloor l_3 \rfloor$$

条件が複数あり分岐が3つ以上になる場合も、1つの実行ステップとして定式化が可能である。

$$l_1 \wedge \alpha : B_1 \supset \alpha = \lfloor l_2 \rfloor,$$
$$l_1 \wedge \alpha : \neg B_1 \wedge B_2 \supset \alpha = \lfloor l_3 \rfloor,$$
$$l_1 \wedge \alpha : \neg(B_1 \vee B_2) \supset \alpha = \lfloor l_4 \rfloor$$

5. (アウェイトによる条件分岐)
「定められた時間以上条件が成立するならば次の実行ステップへ遷移、それ以外はその実行ステップで待ち続ける」という処理をアウェイトと呼ぶ。複数の条件についてアウェイトする場合は、分岐も兼ねることになる。

図 3.4の実行ステップ l は、$0 \leq i \leq m$ $(m \geq 0)$ なる任意の i について条件 C_i が K_i 単位時間成立するならば次の実行ステップ l_i に実行が移るならば、l のプログラム公理は以下

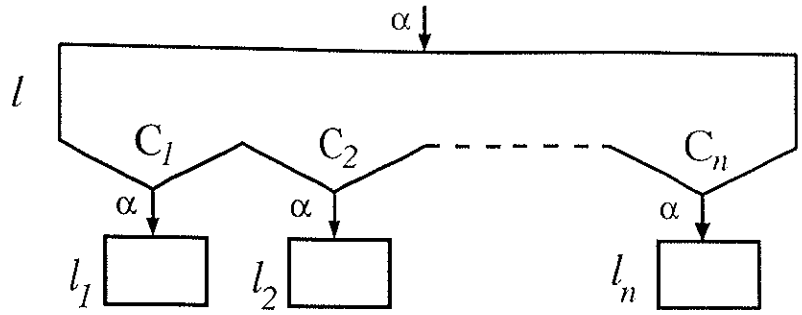


図 3.4: アウェイトによる条件分岐

の通りである (ただし任意の i,j について $i \neq j$ ならば $C_i \supset \neg C_j$ とする)。

$$l \supset \bigwedge_{0\leq i\leq m}(K_i \leq \lfloor \neg C_i \rfloor \supset \alpha \leq K_i \wedge \alpha = \lfloor l_i \rfloor) \\ \wedge((\bigwedge_{0\leq j\leq m} \neg C_j) \supset (\bigvee_{0\leq k\leq m} \lfloor C_k \rfloor \leq \alpha))$$

特に、分岐が無く、成立を待たなければならない条件が無い ($m = 0$, $C_0 = \text{true}$) 場合は

$$l \supset \alpha \leq K_0 \wedge \alpha = \lfloor l_0 \rfloor$$

となる。これは l の実行時間の上限と (単純な) 遷移を表すプログラム公理となっている。

6. (プログラム変数と代入文)
全てのプログラム変数は、「プログラム変数の値は代入 (通信による値の代入を含む) 以外では変化しない」という保存性の公理 [16] を満たすものとする。このことは、TA では、「プログラム変数の値が変化するときはいずれかの代入を実行するブロックで拍車が駆動したときである」というプログラム公理として定式化される。たとえば α 系、β 系の二つの並行プロセスから構成されるプログラム系 $\mathbf{P} == \{\mathbf{P}_\alpha || \mathbf{P}_\beta\}$ 中[2]で大域変数 J について

(a) α 系のプロセスのラベル l_{10} で表されるステップで初期値を代入される

(b) α 系のプロセスのラベル l_{12} で表されるステップと β 系のプロセスのラベル l_{22} で表されるステップでそれぞれ計算が行なわれ J の値が変更される

ものとすると、J に関する保存性の公理は次のプログラム公理として表される。

$$\forall x(J = x \supset \lfloor J \neq x \rfloor = \lfloor l_{10} \rfloor; \alpha \\ \vee \lfloor J \neq x \rfloor = \lfloor l_{12} \rfloor; \alpha \\ \vee \lfloor J \neq x \rfloor = \lfloor l_{22} \rfloor; \beta)$$

これは「今プログラム変数 J の値が x であるならば、J の値が x でなくなるのは l_{10}, l_{12} で α が駆動したときか、l_{22} で β が駆動したときである。」ことを示す。

複数のプロセスで共有される大域変数の保存性の公理には複数の種類の拍車が現れ、そうでない局所変数の保存性の公理にはただ一つの拍車しか現れない。

プログラム変数への代入は有理変数を媒介に定式化する。たとえば l_1 のステップでプログラム変数 J がインクリメントされることは rational variable r を用いて次のように定式化される。

$$\forall r(l_1 \wedge \lfloor J = r \rfloor \supset \alpha : J = r + 1)$$

[2] "||" は並行動作を表す。

これと J に関する保存性の公理から

$$\forall r(l_1 \wedge \lfloor J = r \rfloor \supset \alpha = \lfloor J = r+1 \rfloor)$$

が導ける。

定数の代入については、有理変数の媒介は不要となる。たとえば l_2 のステップで J に定数 k が代入されることは次のように定式化される。

$$l_2 \supset \alpha : J = k$$

以上により流れ図型プログラムについて半自動的な定式化が可能である。こうして定式化された各実行ステップのフェアネスからプログラム系全体のフェアネスを証明する。

3.1.3 プロセスの同期の表現

CSP(Communicating Sequential Processes[6]) の重要な概念であるランデブー (Rendezvous) を TA で表現する。ランデブーとは 2 つのプロセスが同期することであり、TA では以下のようなプログラム公理で表すことができる。例として送受信を伴うランデブーを考える。送信 (受信) する $\alpha(\beta)$ 駆動プロセス $P_\alpha(P_\beta)$ の同期を行なうブロックのプログラムラベルを $A(B)$ とする。また、同期に伴いプロセス P_α のプログラム変数 x の値がプロセス P_β のプログラム変数 y に送信されるとする。CSP では、

$$\begin{aligned} P_\alpha &: P_\beta!y, \\ P_\beta &: P_\alpha?x \end{aligned}$$

と表される。P_α, P_β のブロックを A, B とすると TA のプログラム公理で

$$\begin{aligned} \lfloor A \rfloor;\alpha &= \lfloor B \rfloor;\beta, \\ \lfloor A \rfloor : x = c &\supset \lfloor \beta \rfloor : y = c \end{aligned}$$

と表さる。1 行目は同期を、2 行目は送受信を表している。

3.2 Tense Arithmetic に基づく検証手法 Automata Spur

TA を用いて並行プログラム系検証を行なうとき、前節で述べたように対象プログラムを機能ブロックに分け、それらを拍車をもとにプログラム公理として定式化する。拍車の駆動する系列をグラフオートマトンの入力と見なし、更に各実行ステップに実行時間の上限や下限を割り当て、動作を解析する。これによって以下の利点が得られる。

1. タイミングチャート、状態遷移図を用いた従来の手法を厳密化した手法であるため、それらの知見をとり入れることができる。

2. 拍車によって計算実行列をインデックス付け、その枝刈りを容易にする。

拍車の駆動する系列によって、システム全体の挙動 (各構成プロセスのそれと比べて複雑でありしばしば理解が困難) についての場合分けが容易であり、たとえば到達可能な状態と不可能な状態の判別に有効である。

3. 実行時間の幅の伝播の表現・解析が容易である。

実行時間を含む証明では、実行時間に関する制約が増えるため可能な条件分岐が少なくなることで証明が簡単になることが多いことを確認し、純粋に論理的に検証を行なう場合よりも有効である。

拍車の駆動のインデックスづけには、駆動する順序関係にのみ注目したリスト表記[3]を用いる。このリストを初期拍車列と呼ぶ。複数の拍車が同時に駆動する場合は、それらを( )で括って表す。

たとえば、並行プログラム系 P が 2 つのプロセス $\mathbf{P}_\alpha, \mathbf{P}_\beta$ から構成されそれぞれ拍車 α, β によって駆動する場合、ある適当な初期状態 Θ からの拍車の駆動する順番が α, β, α と β(同時) ,α であることを初期拍車列

$$\langle \alpha, \beta, (\alpha, \beta), \alpha \rangle$$

で表す。 tense formula で記述すれば

$$\begin{aligned} \Theta \supset\ & \alpha < \beta \\ & \wedge\ \alpha\beta < \alpha\alpha \\ & \wedge\ \alpha\beta\alpha = \alpha\beta\beta \\ & \wedge\ \alpha\beta\alpha\alpha < \alpha\beta\alpha\beta \end{aligned}$$

となる。拍車の 1 回の駆動~ 3 回の駆動までしか注目しない場合は初期拍車列

$$\begin{aligned} &\langle \alpha \rangle, \\ &\langle \alpha, \beta \rangle, \\ &\langle \alpha, \beta, (\alpha, \beta) \rangle \end{aligned}$$

で表される。

3.3 他体系との比較

並行プログラム系における実時間問題の検証のためには様々な体系が提唱されている。時相論理 [18] を拡張した実時間時相論理 (Real-Time Logic, RTL, [42]), 区間時相論理 (Interval Temporal Logic, ITL, [26, 27]) 区間時相論理の発展 dulation calculus[2], linear hybrid automaton[1, 20, 40, 41] などであり、我々のグループでも ν- 理論 [8, 38], エンヴェロープ理論 [31, 32, 13, 30], SOFA[12, 24] を研究している。これらの多くは連続的に変化する外部現象を離散的に制御するプログラム (reactive system[22]) の検証を目的とする。 TA は連続的に変化する外部現象を扱うようには構成されていないが、拡張は可能である。

3.3.1 時相論理を基礎とする体系との比較

時相論理は自然数時間を扱う体系である。時間によって真理値の変化する命題 F について

1. $\bigcirc F$ 「次に F が成立する (next)」

2. $\Diamond F$ 「いつかは F が成立する (possibility, eventuality)」

[3]⟨ ⟩ で括ったものとする。

3. $\Box F$ 「常に F が成立する (necessity)」

といった表現が出来る。$\Diamond F$ と $\Box F$ には、

$$\Diamond F \equiv \neg\Box\neg F, \quad \Box F \equiv \neg\Diamond\neg F$$

という関係がある。

「F_1 が成立するまで F が成立し続ける」という概念を表すために until 演算子 ($\mathcal{U}$) を導入し $F\ \mathcal{U}\ F_1$ と記述する。until 演算子を基本的な演算子とすることによって $\Box$ と $\Diamond$ を定義することができる。

$$\Box F \stackrel{\text{def}}{\equiv} F\ \mathcal{U}\ \text{false}$$

時相論理は next 演算子と until 演算子から構成することができる。$\bigcirc$, $\Diamond$, $\Box$ という表現は、時相論理を基礎とする体系において有理数時間や実数時間上に拡張され多用されている。

時間に関する制約がある仕様を、具体的な時間値を用いて論理的に記述し、検証したいという要求は強く、時相論理は RTL や ITL として有理数時間や実数時間へ拡張された。ただし時相論理は一度に一つのプロセスしか実行されないモデルを対象にしているために本質的には一つのプロセッサでインターリーブに動作するものしか扱えないことが問題となる。

RTL は、時相論理について時間を有理数時間や実数時間とし、実際の時間値と命題について以下の記号を追加したものである。RTL は時相演算子として通常の時相論理の until 演算子と、実時間演算子として *at* と *for* を持ち

1. F *at* 1 hour 「1 時間後に F が成立する。」

2. F *for* 1 hour 「現在から 1 時間後まで F が成立し続ける。」

という表現を可能とする。前者では時間によって変わる F の真理値を得る。後者は、ある時間まで (この場合 1 時間後) 区切った期間について $\Box F$ が成立するという、necessity 演算子の拡張になっている。

ITL は実数時間を扱う体系である。時間は有限個の区間 (interval) から構成され、区間の長さは非負実数値となる。長さ 0 の区間と正の長さの区間が混在する。区間の時間の長さを表す特別な記号 l (length of interval) を用いる。有限の長さの区間を有限個しか扱えないため、停止しないプログラムの証明に問題がある。

TA は有理数時間を扱う体系であるが実数時間への拡張も検討されている。また TA 上で時相論理の大部分は展開可能である。TA の tense formula において時相論理的表現は tense "$\lfloor\ \rfloor$"、経過演算子 ";"、tense term 同士の (時間的) 比較 "$=$", "$\leq$" によってもたらされる。

時相論理の論理式 $\Diamond F$, $\Box F$ は、TA では限定記号を用いれば $\exists r(r : F)$, $\forall r(r : F)$ と表現される。特に F が時相演算子 (temporal operator) の出現しない論理式 P の場合、$\Diamond P$, $\Box P$ は TA では $T_C^{\mathcal{Q}}$ の論理式 P に対して $\lfloor P \rfloor < \infty$, $\lfloor \neg P \rfloor = \infty$ と表される。命題が立ち上がるまでの時間を比較する $\leq$ は時相論理の until 演算子の性質を包含する。TA のシーケント

$$F_1, \ldots, F_m \ \rightarrow\ G_1, \ldots, G_n$$

は時相論理の

$$\Box(F_1 \wedge \ldots \wedge F_m \ \supset\ G_1 \vee \ldots \vee G_n)$$

に対応する。$\bigcirc F$ は、検証対象の並行プログラム系が拍車 γ_0, γ_1, $\cdots$ から構成される場合、

$$\gamma_0 : F \vee \gamma_1 : F \vee \cdots$$

で表される。実際のプログラム検証においては、漠然と次の状態を表す next 演算子 ($\bigcirc$) よりも、プロセスの識別を同時に行なえる拍車の方が優れている。

このように TA で RTL や ITL の論理式を展開できるが、ただし tense "$\lfloor\ \rfloor$" のアーギュメントにはこれらの記号が出現しない。よって tense formula は時相演算子的表現がネスティングしない範囲の表現に限られ、TA のシーケント

$$F_1, \ldots, F_m \ \rightarrow\ G_1, \ldots, G_n$$

について $F_1, \ldots, F_m, G_1, \ldots, G_n$ のそれぞれで時間論理的表現がネスティングしないため、シーケント全体としては高々 1 段しかネスティングがない。プログラム検証においてはこの範囲の論理式で十分であり、時相演算子のネスティングは、限定的にしか必要ではなかった [15]。

duration calculus は ITL を拡張したものであり、区間の中で命題が成立する時間の総和を積分の形で得られる体系である。ITL では特別な記号 l で得ていた区間の長さを、$\int \mathrm{true}$ で定義することができる。duration caluculus は、ITL と同じく時間を有限個の有限区間として扱うため、OS などの停止しないプログラムを扱うには不自然であり問題が生じる。また、たとえば duration caluculus で「ガスが洩れているならばそれは 1 単位時間以下 (の区間) しか続かない」という仕様は

$$\lceil Leak \rceil \ \Rightarrow\ l \leq 1$$

と表現される。これは TA のシーケント

$$Leak \ \rightarrow\ \lfloor \neg Leak \rfloor \leq 1$$

と表現される。duration caluculus で「ガスが洩れるのは 30 単位時間以上間隔がある」という仕様は

$$\lceil Leak \rceil ; \lceil \neg Leak \rceil ; \lceil Leak \rceil \ \Rightarrow\ l \geq 30$$

と表現される。これは TA のシーケントで

$$Leak \rightarrow 30 \leq \lfloor \neg Leak \rfloor ; \lfloor Leak \rfloor$$

と表現される。区間という概念を意識せずとも似た表現になる。

liner hybrid automaton は拡張したオートマトンによって reactive system を記述する。外界の対象の連続的な変化はオートマトンの各状態において外部の変数として観察され、それに対しするプログラムでの制御を含む離散的な変化は状態遷移で表される。一部自動検証を行なうことができるが、対象がプログラムそのものではなくオートマトンであることが問題となる場合がある。たとえば、図 3.5 の hybrid automaton は状態 state_1 で条件 $\mathrm{condition}_1$ が成立するならば制御プログラムでは action_1 (制御変数への代入等) を行ない次の状態 state_2 に遷移する。$\mathrm{condition}_1$ が成立しなければ state_1 のままであり、外界の状態は 1 つの状態に留まる限り連続で変化する。状態 state_2 も同様とする。 state_1 で $\mathrm{condition}_1$ が成立しているとき、無限小時間で action_1 が行なわれ state_2 に遷移する。ここで state_2 に遷移した瞬間、条件 $\mathrm{condition}_2$ も成立していたならば同じ瞬間に action_2 も行なわれ state_3 に遷移することに注意しなければならない。

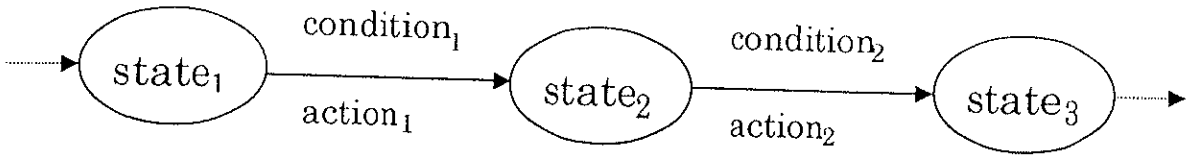


図 3.5: liner hybrid automaton

すなわち state_1 から、state_2 に留まらず直接に state_3 へ遷移することになり、オートマトンとしては普通だがプログラムとしては不自然な動作となる。記述の方法によっては action_1 と action_2 の内容により矛盾した動作を表すこともある。プログラムを liner hybrid automaton として記述して検証する際には注意が必要である。これに対し TA では、オートマトンでなくプログラムを定式化するので特別な注意は不要である。プログラム公理としては、各状態をアウェイトとして扱い、

$$\begin{aligned}
\text{state}_1 \supset\ & \alpha = \lfloor \text{state}_2 \rfloor \\
& \wedge\ \alpha : \text{action}_1 \\
& \wedge\ (\text{condition}_1 \supset \alpha < \neg\lfloor \text{condition}_1 \rfloor) \\
& \wedge\ (\neg\text{condition}_1 \supset \lfloor \text{condition}_1 \rfloor \leq \alpha), \\
\text{state}_2 \supset\ & \alpha = \lfloor \text{state}_3 \rfloor \\
& \wedge\ \alpha : \text{action}_2 \\
& \wedge\ (\text{condition}_2 \supset \alpha < \neg\lfloor \text{condition}_2 \rfloor) \\
& \wedge\ (\neg\text{condition}_2 \supset \lfloor \text{condition}_2 \rfloor \leq \alpha)
\end{aligned}$$

と表す。state_1 から state_3 へは拍車 α が 2 回駆動するため $(\text{state}_1 \supset \alpha\alpha = \lfloor \text{state}_3 \rfloor)$、必ず一旦 state_2 を経由する。もし「state_1 で condition_1 かつ condition_2 が成立するならば state_3 へ直接遷移する」場合があるならばその遷移をプログラム公理に補う。扱う対象によるがプログラム検証の立場では、TA の方がより自然にプログラムの定式化を行なえる。

プログラム検証中に扱う時間値が現われる性質については有理数理論に帰着できる証明は多く、それを自然に実現するよう厳密に体系づけられた TA はこれらのいずれの体系と比べても有効であると考える。

3.3.2 ν- 理論を基礎とする体系

ν- 理論は我々のグループの研究全ての基礎となる理論である。変数全てを高階にして時間の関数として扱い、複数個のプロセッサによって動作するプログラムについても検証可能であり、その上で時相論理を展開することもできる強力な体系である。

エンヴェロープ理論は、体系としては命題をそれが成立する時間の集合として扱い、ブール束をもとに構築したものである。体系としては簡潔で、証明を代数的に行なうが、モノジェニック、モノクロージャーという概念を把握していなければならない。TA ではこの問題を、命題を、成立する時間の集合ではなく「最初に成立する時刻」として解釈する事により回避している。TA における命題 P の立ち上がり $\lfloor P \rfloor$、経過演算子を用いた $\lfloor P \rfloor; \lfloor P_1 \rfloor$ という表現はエンヴェロープ理論ではエンヴェロープと呼ぶ演算子 / (命題が成立する時間全体の集合について、一種の閉包を得る) を用いた表現 $P/$、$(P/\cap P_1)/$ に対応する。$(P/\cap P_1)/$ は略記として $P; P_1$ と表される。に対応する。エンヴェロープ理論では、「P が成立するのは P_1 が成立する以前である。」とは

$$P/ \preceq P_1/$$

と表される[4]。エンヴェロープ理論では常に、命題が真になる時間の集合と、集合演算を意識しなければならない。その分、TA よりも記述力は高いが現在わかっている範囲ではプログラム証明において TA の記述力で十分である。

SOFA は、解析学 FA[37] および解析的意味論に基づき、解析的な検証を遂行する。FA の基礎となる高階の論理 LS を扱わなければならない。以上に対し TA で必要なのは有理数理論についての知識のみであり、数学的レベルで行なわれたプログラム検証について少ない労力で (もとの証明に近い形で) 論理的な裏付けをつけることができる。

エンヴェロープ理論や SOFA で連続系の制御システムを検証対象とする拡張は進められている。なお Automata Spur によるプログラムの定式化・検証手法は、SOFA への適用も容易である。

[4]包含関係の記号を用いれば $P/ \supseteq P_1/$ である。

第 4 章

並行プログラム系の検証例

この章では TA を用いた並行プログラム系の実時間問題の検証例を示す。例題 1, 2 については TA の原形となった体系で実時間値を扱った証明が [34, 35] で、TA によるものが [36] でそれぞれ発表されている。例題 3 については数学的手法での証明が [33] 発表されている。例題 3 については概要のみ示す。

4.1 Dekker の解

危険領域を持つ並行プログラム系 Dekker の解 [3] を考える。このプログラムは、飢餓回避 (starvation free)、停頓回避 (deadlock free)、相互排反 (mutual exclusion) の性質を持ち、OS の基礎となる機能を備えている。本論文では、飢餓回避の証明を行なうだけでなく、各プロセスの最大待ち時間を解析する。Dekker の解を抽象化したものを図 4.1に示す[1]。ここでは、2 つの並行プロセスから構成されているが、n 個の並行プロセスへの拡張は容易である。

この並行プログラム系は拍車 α で駆動する P_{left}, β で駆動する P_{right} の 2 つのプログラムからなり、共通のプログラム変数 T_L を持つ。L_L, R_L は分岐を兼ねたアウェイト (条件成立まで待ち)、M_L, M_R は危険領域、P_L, P_R はプログラム本体、S_L, S_R はアウェイトである。T_L は左側のプロセスが優先であることを示す。

ここでは、各プロセスに適切な動作時間を与えることにより検証を行なう。読み易くするため、具体的な数値例にする。$M_L(M_R)$ は 90(70) 単位時間から 120(130) 単位時間の間、$P_L(P_R)$ は 90(70) 単位時間以上の動作時間とする。また、各アウェイトは、0.9(0.7) 単位時間の間条件が成り立ち続けるならば、条件が偽になる前に遷移するものとする。

4.1.1 プログラム公理

Dekker の解のプログラム公理を以下に示す。ラベルの保存性[2]や排反性[3]については省略する。

(P_{left} のプログラム公理)

---

[1] これが元のプログラム系の本質的な性質を全て保存していることは軌跡準同型 [23] によって証明されている [11]。

[2] ラベル L_L は対応する拍車 α が駆動するまでは値が保存される。

$$L_L \supset \alpha = \lfloor \neg L_L \rfloor$$

$M_L P_L, S_L$ や β 系の $L_R, M_R P_R, S_R$ についても同様。

[3] $L_L, M_L P_L, S_L$ は互いに排反である。β 系の $L_R, M_R P_R, S_R$ についても同様。

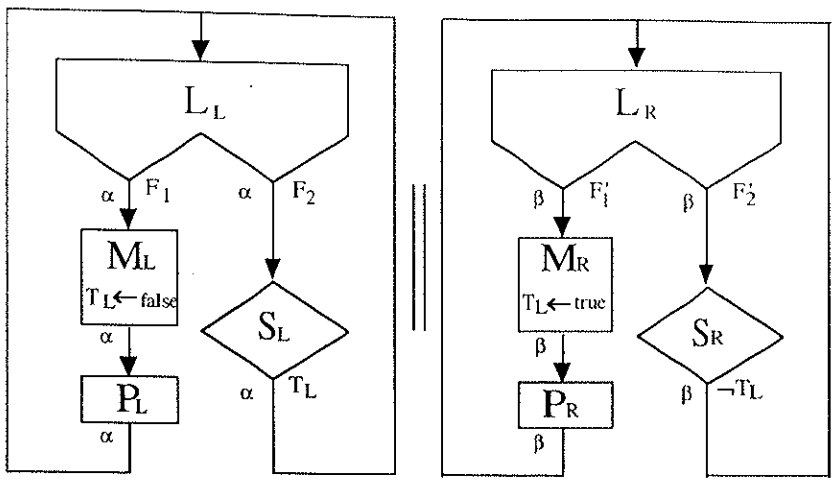


図 4.1: Dekker の解

$$
\begin{aligned}
L_L \ &\supset \ ((0.9 \le \lfloor \neg F_1 \rfloor \ \supset \ \alpha < 0.9 \ \wedge \ \alpha = \lfloor M_L \rfloor) \\
&\quad \wedge (0.9 \le \lfloor \neg F_2 \rfloor \ \supset \ \alpha < 0.9 \ \wedge \ \alpha = \lfloor S_L \rfloor) \\
&\quad \wedge (\neg F_1 \ \wedge \neg F_2 \ \supset \ \lfloor F_1 \rfloor \le \alpha \ \vee \ \lfloor F_2 \rfloor \le \alpha)), && (4.1) \\
M_L \ &\supset \ \alpha < 120 \ \wedge \ \alpha = \lfloor P_L \rfloor \ \wedge (\alpha : \neg T_L), && (4.2) \\
\neg M_L \ &\supset \ \lfloor M_L \rfloor : 90 \le \alpha, && (4.3) \\
\neg P_L \ &\supset \ \lfloor P_L \rfloor : 90 \le \alpha, && (4.4) \\
S_L \ &\supset \ ((0.9 \le \lfloor T_L \rfloor \ \supset \ \alpha \le 0.9 \ \wedge \ \alpha = \lfloor L_L \rfloor) \\
&\quad \wedge (\neg T_L \ \supset \ \lfloor \neg T_L \rfloor \le \alpha)), && (4.5) \\
L_L \ &\vee \ M_L \ \vee \ S_L \ \vee \ P_L, && (4.6) \\
T_L \ &\supset \ \lfloor \neg T_L \rfloor = \lfloor M_L \rfloor ; \alpha, && (4.7) \\
& && (4.8)
\end{aligned}
$$

(P_{right} のプログラム公理)

$$
\begin{aligned}
L_R \ &\supset \ ((0.7 \le \lfloor \neg F'_1 \rfloor \ \supset \ \beta < 0.7 \ \wedge \ \beta = \lfloor M_R \rfloor) \\
&\quad \wedge (0.7 \le \lfloor \neg F'_2 \rfloor \ \supset \ \beta < 0.7 \ \wedge \ \beta = \lfloor S_R \rfloor) \\
&\quad \wedge (\neg F'_1 \ \wedge \neg F'_2 \ \supset \ \lfloor F'_1 \rfloor \le \beta \ \vee \ \lfloor F'_2 \rfloor \le \beta)), && (4.9) \\
M_R \ &\supset \ \beta < 130 \ \wedge \ \beta = \lfloor P_R \rfloor \ \wedge (\beta : T_L), && (4.10) \\
\neg M_R \ &\supset \ \lfloor M_R \rfloor : 70 \le \beta, && (4.11) \\
\neg P_R \ &\supset \ \lfloor P_R \rfloor : 70 \le \beta, && (4.12) \\
S_R \ &\supset \ ((0.7 \le \lfloor T_L \rfloor \ \supset \ \beta \le 0.7 \ \wedge \ \beta = \lfloor L_R \rfloor) \\
&\quad \wedge (T_L \ \supset \ \lfloor \neg T_R \rfloor \le \beta)), && (4.13) \\
L_R \ &\vee \ M_R \ \vee \ S_R \ \vee \ P_R, && (4.14) \\
\neg T_L \ &\supset \ \lfloor T_L \rfloor = \lfloor M_R \rfloor ; \beta, && (4.15)
\end{aligned}
$$

ただし

$$\begin{aligned} F_1 &\stackrel{\text{def}}{\equiv} \neg(L_R \vee M_R), \\ F_2 &\stackrel{\text{def}}{\equiv} (L_R \vee M_R) \wedge \neg T_L, \\ F_1' &\stackrel{\text{def}}{\equiv} \neg(L_L \vee M_L), \\ F_2' &\stackrel{\text{def}}{\equiv} (L_L \vee M_L) \wedge T_L. \end{aligned}$$

4.1.2 最大待ち時間の解析

プログラムのコントロールが M_L もしくは M_R に到達しないことを飢餓状態と呼ぶ。プロセスが危険領域に入るまでの最大待ち時間を形式的に解析することにより飢餓回避を証明する。

飢餓回避の本質は初期状態 $\Theta \stackrel{\text{def}}{\equiv} L_L \wedge S_R \wedge \neg T_L$ から M_L に到達するか否かに帰着できる。

プログラム公理 (4.1)(4.12) より、

$$\begin{aligned} L_L \wedge S_R \wedge \neg T_L \supset\ & (0.9 \leq \lfloor \neg F_1 \rfloor \supset \alpha < 0.9 \wedge \alpha = \lfloor M_L \rfloor) \\ & \wedge (0.7 \leq \lfloor T_L \rfloor \supset \beta \leq 0.7 \wedge \beta = \lfloor L_R \rfloor) \end{aligned} \tag{4.16}$$

が導かれる。

ラベルの保存性と (4.14) より

$$L_L \supset \alpha \leq \lfloor \neg F_2' \rfloor, \tag{4.17}$$

$$S_R \wedge \neg T_L \supset \beta \leq \lfloor \neg F_1 \rfloor \tag{4.18}$$

が導かれ、さらに

$$\begin{aligned} L_L \wedge S_R \wedge \neg T_L \supset\ & ((\alpha \leq \lfloor \neg T_L \rfloor \vee \beta \leq \lfloor \neg T_L \rfloor)) \\ & \wedge \beta \leq \lfloor \neg F_1 \rfloor \end{aligned} \tag{4.19}$$

が導かれる。(4.16)(4.19) より

$$\begin{aligned} L_L \wedge S_R \wedge \neg T_L \supset\ & ((0.9 \leq \beta \supset \alpha < 0.9) \\ & \wedge (0.7 \leq \alpha \supset \beta < 0.7)). \end{aligned} \tag{4.20}$$

(4.20) と定理 (2) 全順序性を用いて

$$\begin{aligned} \Theta \supset\ & ((\alpha \leq \beta \supset \alpha < 0.7) \\ & \wedge (\beta < \alpha) \supset \beta < 0.7) \end{aligned} \tag{4.21}$$

が導ける。最初の拍車の駆動については

case 1 $\beta < \alpha$ の場合 $\beta \leq 0.7$

case 2 $\alpha < \beta$ の場合 $\alpha < 0.7$

case 3 $\alpha = \beta$ の場合 $\alpha < 0.7 \wedge \beta < 0.7$

の3つの場合が考えられる。それぞれの場合の計算実行列を表4.1 ～4.3にまとめた。

spurs from 0 の欄は初期拍車列すなわち初期状態以後の拍車の駆動する履歴を、α-OK (β-OK) の欄は、$P_{left}(P_{right})$ が次の状態に遷移するための必要条件をそれぞれ表す。

初期状態 Θ では定義により F_1, F_2' が成立し、F_2, F_1' は成立していない。

最初に β が駆動する case 1 については、途中の状態から

$$L_L \wedge L_R \wedge \neg T_L \supset \alpha < 0.9, \tag{4.22}$$

$$S_L \wedge L_R \wedge \neg T_L \supset \beta < 0.7, \tag{4.23}$$

$$\neg(S_L \wedge M_R \wedge \neg T_L) \supset \lfloor S_L \wedge M_R \wedge \neg T_L \rfloor : 70 \leq \beta < 130, \tag{4.24}$$

$$S_L \wedge P_R \wedge T_L \supset \alpha < 0.9, \tag{4.25}$$

$$L_L \wedge P_R \wedge T_L \supset \alpha < 0.9 \tag{4.26}$$

がそれぞれ導かれる。$S_L \wedge P_R \wedge T_L$ の状態では α も β も共にアウェイトの十分条件を満たす。実時間値を用いない証明においてここでは場合分けを行なって検証しなければならないが、M_R の実行時間が定められているために必ず α の方が先に駆動する。

case 1 において拍車は必ず $\langle \beta, \alpha, \beta, \beta, \alpha, \alpha \rangle$ の順に駆動しプロセス P_{left} が危険領域 M_L に到達する。それまでにかかる時間は経過した各ブロックの実行時間の上限と下限の累計より 70 から 134.1 単位時間の間である。

$$L_L \wedge S_R \wedge \neg T_L \supset 70 \leq \lfloor M_L \rfloor < (0.7 + 0.9 + 0.7 + 130 + 0.9 + 0.9)$$

なお、最初に α が β 以前に駆動する場合は同様の方法で 0 から 0.7 単位時間以内に M_L が実行されることが導ける。即ち、以下の式が成り立つ。

$$\Theta \supset 70 \leq \lfloor M_L \rfloor < 134.1 \vee \lfloor M_L \rfloor < 0.7.$$

以上により、α の駆動が β の駆動以前の場合には開始時刻から 0.7 単位時間以内、β の駆動が α よりも早い場合には開始時刻から 70 単位時間以降 134.1 単位時間以内にコントロールが M_L に到達することが導けた。動作時間を仮定しない場合に比べ、可能な条件分岐が減っているため、証明を簡単に行なうことができた。

また、この条件下では、「各プロセスともに相手のプロセスが2度危険領域に入るのを待つことは無い」ことを表す

$$\neg M_L \supset \lfloor M_L \rfloor \leq 134.1 \leq 70 + 70 + 0.7 \leq \lfloor M_R \rfloor; \lfloor \neg M_R \rfloor; \lfloor M_R \rfloor$$

も証明可能である。

spurs from Θ	time diff. min	max	state α drv.	β drv.	T_L	permission αOK	βOK	spur expect. α	β
$\langle\rangle$			L_L	S_R	false	S_R	$\neg T_L$	$0.9 \leq \lfloor \neg S_R \rfloor$ $\supset \alpha < 0.9$	$0.7 \leq \lfloor T_L \rfloor$ $\supset \beta < 0.7$
$\langle\beta\rangle$		0.7	L_L	L_R	false	$L_R \wedge \neg T_L$	not hold	$0.9 \leq \lfloor \neg(L_R \vee \neg T_L) \rfloor$ $\supset \alpha < 0.9$	$\alpha < \beta \vee$ $\alpha = \beta = \infty$
$\langle\beta\alpha\rangle$		0.9	S_L	L_R	false	not hold	S_L	$\beta < \alpha \vee$ $\alpha = \beta = \infty$	$0.7 \leq \lfloor \neg S_L \rfloor$ $\supset \beta < 0.7$
$\langle\beta\alpha\beta\rangle$		0.7	S_L	M_R	false	not hold	don't care	$\beta < \alpha \vee$ $\alpha = \beta = \infty$	$70 \leq \beta < 130$
$\langle\beta\alpha\beta\beta\rangle$	70	130	S_L	P_R	true	T_L	don't care	$0.9 \leq \lfloor \neg T_L \rfloor$ $\supset \alpha < 0.9$	$70 \leq \beta$
$\langle\beta\alpha\beta\beta\alpha\rangle$		0.9	L_L	P_R	true	P_R	don't care	$0.9 \leq \lfloor \neg P_R \rfloor$ $\supset \alpha < 0.9$	$70 \leq \beta$
$\langle\beta\alpha\beta\beta\alpha\alpha\rangle$		0.9	M_L	P_R	true	don't care	don't care	$90 \leq \alpha < 120$	$70 \leq \beta$

表 4.1: Dekker の解における飢餓回避問題 : case 1. $\Theta \wedge \beta < \alpha$

spurs from Θ	time diff. min	max	state α drv.	β drv.	T_L	permission αOK	βOK	spur expect. α	β
$\langle\rangle$			L_L	S_R	false	S_R	$\neg T_L$	$0.9 \leq \lfloor \neg S_R \rfloor$ $\supset \alpha \leq 0.9$	$0.7 \leq \lfloor T_L \rfloor$ $\supset \beta < 0.7$
$\langle\alpha\rangle$		0.7	M_L	S_R	false	don't care	$\neg T_L$	$90 \leq \alpha < 120$	$0.7 \leq \lfloor T_L \rfloor$ $\supset \beta < 0.7$

表 4.2: Dekker の解における飢餓回避問題 : case 2. $\Theta \wedge \alpha < \beta$

spurs from Θ	time diff. min	max	state α drv.	β drv.	T_L	permission αOK	βOK	spur expect. α	β
$\langle\rangle$			L_L	S_R	false	S_R	$\neg T_L$	$0.9 \leq \lfloor \neg S_R \rfloor$ $\supset \alpha < 0.9$	$0.7 \leq \lfloor T_L \rfloor$ $\supset \beta < 0.7$
$\langle(\alpha \cdot \beta)\rangle$		0.7	M_L	L_R	false	don't care	$M_L \wedge T_L$	$90 \leq \alpha < 120$	$0.7 \leq \lfloor \neg(M_L \wedge T_L) \rfloor$ $\supset \beta < 0.7$

表 4.3: Dekker の解における飢餓回避問題 : case 3. $\Theta \wedge \alpha = \beta$

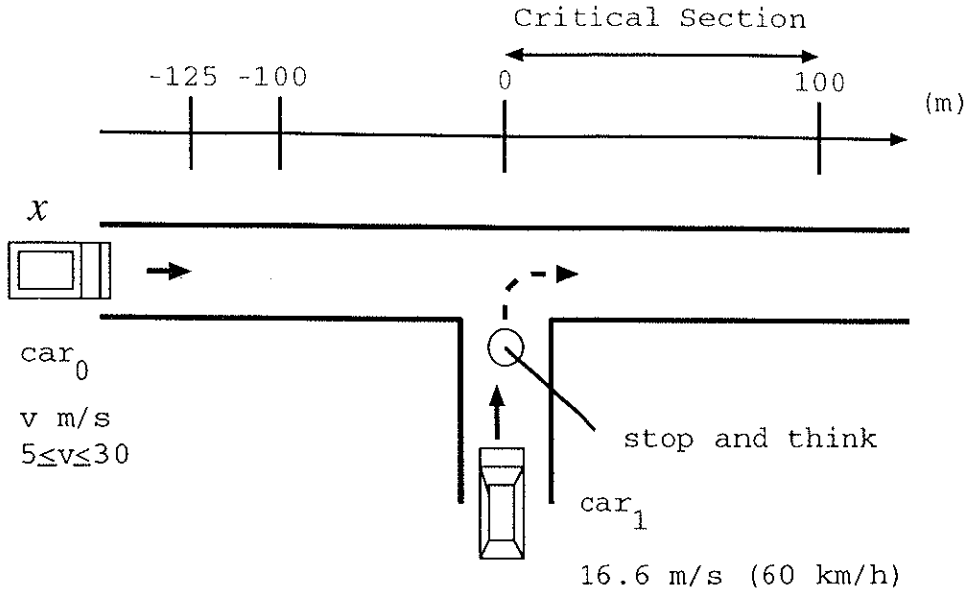


図 4.2: T 字路での合流

4.2 T 字路での合流問題

見通しの悪い T 字路交差点におけるビークルの簡単な合流問題について、与えられた合流条件のもとで 2 台のビークルが衝突する場合があるか否かについて検証を行なった。このような問題では実際の時間の値があってこそ意味のある検証となる。

図 4.2のように交差点を原点、左側を負、右側を正とし 1 次元座標をとる。簡単のために合流される側のビークルは 1 台のみとし、 car_0 と呼ぶ。各ビークルは点と考え、位置 (単位は m) を x、速さ (単位は m/s) は一定で v とする。ただし、 $5 \leq v \leq 30$ とする。合流する側のビークル (car_1 と呼ぶ) は T 字路の直前で一旦停止し、次のいずれかの条件が成立する時にアクセルを踏み、成立しない時は成立するまで待つ。

1. car_0 が図の左側または右側に十分離れている。すなわち $y < -125$ または $100 < y$
2. car_0 が交差点から右側に $100m$ 以内の位置にいて、 car_1 に追突されない程度に速度が速い。すなわち $0 < y < 100 \wedge 12.5 < v$
3. car_0 が交差点から左側に $100m$～$125m$ の位置にいるが、 car_1 に追突しない程度に速度が遅い。すなわち $-125 \leq y \leq -100 \wedge v < 25$

car_1 がアクセルを踏んだ後、実際に動き出すまでに 2 秒間かかり (この間はまだ交差点に進入しておらず衝突しない)、 2 秒後に交差点に進入し、一定の速さ $16.\dot{6}m/s$ で右方向に進むものとする。すなわち、 car_1 はアクセルを踏むタイミングだけを制御する。最初に car_0 は交差点から負の方向 (左側) に十分離れた位置にあり、 car_1 も交差点に到達していない状態とする。それ以後有限時間内に car_0 が $-125m$ の地点に、 car_1 は交差点に、それぞれ到達するものとする。この条件のもとで交差点から右に $100m$ の範囲 ($[0, 100]$、 この問題における危険領域) で衝突しないように合流したい。

この問題を TA で扱うために並行プログラム系として捉える (図 4.3)。すなわち 3 つのブロック L_0, L_1, L_2 からなる拍車 α 系のプロセスで car_0 を、同様に 3 つのブロック R_0, R_1, R_2 からな

る拍車 β 系のプロセスで car_1 を表す。各ブロックは、同じ名前のラベルが 1 対 1 で対応づけられるものとする。L_1, L_2 の実行にかかる時間は各々 $100/v$ 秒、R_1 ではアクセルを踏む条件が成立してから 2 秒後に R_2 が実行され始め、R_2 の実行にかかる時間は 6 秒とする。

ブロックと車の位置は次に示す対応づけがされている。

L_0	$\Leftrightarrow$	$x \notin [-125, 100]$	car_0 は交差点から十分離れている (交差点からは見えない)
L_1	$\Leftrightarrow$	$x \in [-125, 0)$	car_0 は交差点左側に接近
L_2	$\Leftrightarrow$	$x \in [0, 100]$	car_0 は危険領域中

R_0	$\Rightarrow$	$y \notin [0, 100]$	car_1 は交差点に未到達
R_1	$\Rightarrow$	$y \notin [0, 100]$	car_1 は交差点に到達。条件によりアクセルを踏む
R_2	$\Leftrightarrow$	$y \in [0, 100]$	car_1 は危険領域中

4.2.1 プログラム公理

プログラム公理は以下の通りである (ラベルの保存性・排反性については省略する)。v は一定であるので $T^{\mathcal{Q}}$ の定数とする。

$$L_0 \supset \alpha < \infty \wedge \alpha = \lfloor L_1 \rfloor, \tag{4.27}$$

$$L_1 \supset \alpha = \lfloor L_2 \rfloor, \tag{4.28}$$

$$\neg L_1 \supset \lfloor L_1 \rfloor : \alpha = 125/v, \tag{4.29}$$

$$L_2 \supset \alpha < 100/v \wedge \alpha = \lfloor L_0 \rfloor, \tag{4.30}$$

$$L_0 \vee L_1 \vee L_2, \tag{4.31}$$

$$R_0 \supset \beta < \infty \wedge \beta = \lfloor R_1 \rfloor, \tag{4.32}$$

$$R_1 \supset \beta = \lfloor R_2 \rfloor, \tag{4.33}$$

$$\begin{aligned} \neg R_1 \supset \lfloor R_1 \rfloor : (&\beta = \lfloor L_0 \rfloor; 2 \vee \\ &100/v < \lfloor \neg L_1 \rfloor \wedge \beta = 2 \vee \\ &\beta = \lfloor L_2 \wedge 12.5 < v \rfloor; 2), \end{aligned} \tag{4.34}$$

$$R_2 \supset \beta = 6 \wedge \beta = \lfloor R_0 \rfloor, \tag{4.35}$$

$$R_0 \vee R_1 \vee R_2. \tag{4.36}$$

アクセルを踏む条件 go の定義は次の通りである。

$$\begin{aligned} go \stackrel{\text{def}}{=} \quad & L_0 \vee L_1 \wedge 100/v < \lfloor \neg L_1 \rfloor \\ & \wedge v < 25 \vee L_2 \wedge 12.5 < v \end{aligned}$$

TA の構文上の制限 ("$\lfloor\ \rfloor$" のアーギュメントは $T_C^{\mathcal{Q}}$ の論理式に限られる) があり、R_1 のプログラム公理中として

$$\neg R_1 \supset \lfloor R_1 \rfloor : \beta = go; 2$$

とは表記できないため異なった形で用いられている。

初期状態 Θ を

$$\Theta \stackrel{\text{def}}{=} L_0 \wedge R_0$$

とし、vの値の範囲を$5 \leq v \leq 30$とする。相手の車および動き出した後の自車の速さは一定なので、ある観察時刻から後、最初の危険領域で衝突しないのは、危険領域内で追い越しが無い、すなわち互いの危険領域に入った時刻と出た時刻が一致せずかつ順序が同じである場合である。衝突の発生 $crush$ を次のように定式化する。

$$\begin{aligned} crush \stackrel{\text{def}}{=} \quad & \lfloor L_2 \rfloor = \lfloor R_2 \rfloor \\ & \vee \lfloor L_2 \rfloor ; \lfloor L_0 \rfloor = \lfloor R_2 \rfloor ; \lfloor R_0 \rfloor \\ & \vee \lfloor L_2 \rfloor < \lfloor R_2 \rfloor \ \wedge \ \lfloor R_2 \rfloor ; \lfloor R_0 \rfloor < \lfloor L_2 \rfloor ; \lfloor L_0 \rfloor \\ & \vee \lfloor R_2 \rfloor < \lfloor L_2 \rfloor \ \wedge \ \lfloor L_2 \rfloor ; \lfloor L_0 \rfloor < \lfloor R_2 \rfloor ; \lfloor R_0 \rfloor \end{aligned}$$

4.2.2 検証

初期状態 $L_0 \wedge R_0$ 以後、car_0 と car_1 が危険領域で衝突しない、すなわち拍車 α, β 各々が3回駆動する間に $crush$ が成立する場合があるか否かを解析した。

まずプログラム公理を一切考慮せず、α, β 各々が3回駆動する初期拍車列 (3.2節) を正則表現的に生成すると以下の63通りが得られる。初期条件を満たす状態から α, β が立ち上がる順番の系列を考えると最初の6回の拍車の立ち上がりで $crush$ を満たす可能性があるのはこの63通りだけである。

$$\begin{gathered} \langle \alpha, \alpha, \alpha, \beta, \beta, \beta \rangle, \\ \langle \alpha, \alpha, \beta, \alpha, \beta, \beta \rangle, \\ \langle \alpha, \alpha, \beta, \beta, \alpha, \beta \rangle, \\ \langle \alpha, \alpha, \beta, \beta, \beta, \alpha \rangle, \\ \langle \alpha, \alpha, \beta, \beta, (\alpha, \beta) \rangle, \\ \langle \alpha, \alpha, \beta, (\alpha, \beta), \beta \rangle, \\ \cdots \\ \langle (\alpha, \beta), (\alpha, \beta), (\alpha, \beta) \rangle. \end{gathered}$$

これらの初期拍車列にについて検証するために簡単なツール (プログラム) を作成して解析した。表4.4はツールによるチェックの出力である。"spurs from 0"の欄は初期拍車列 (表示の都合により a で α を、b で β を表す。リストを表す $\langle\ \rangle$ やコンマは省略する) を、"a2=b2, a3=b3, 交差"の欄はそれぞれ初期状態から観察して

- $\alpha\alpha = \beta\beta$
- $\alpha\alpha\alpha = \beta\beta\beta$
- $\alpha\alpha < \beta\beta$ かつ $\alpha\alpha\alpha > \beta\beta\beta$、または $\alpha\alpha > \beta\beta$ かつ $\alpha\alpha\alpha < \beta\beta\beta$、

のいずれかが成立する初期拍車列について"O"の印が表示される。その右側の6カラムは、初期拍車列で示される拍車の駆動の時間を早い順に左から示す。右側の欄外には、その拍車列での v の範囲についての条件が表示されている。

まず64通りの初期拍車列のうち34通りは $crush$ が成立しない。

$crush$ の各ラベルの tense を拍車を用いて表すと次式になる。

$$\begin{aligned} \Theta \ \supset \ \text{crush} \equiv & \ \alpha\alpha = \beta\beta \\ & \vee \ \alpha\alpha\alpha = \beta\beta\beta \\ & \vee \ \alpha\alpha < \beta\beta \ \wedge \ \beta\beta\beta = \alpha\alpha\alpha \end{aligned}$$

$$\vee\ \beta\beta < \alpha\alpha\ \wedge\ \alpha\alpha\alpha = \beta\beta\beta$$

64 通りの初期拍車列のうち 34 通りは表 4.4 中に "0" の印が表示されていない。すなわち、これらの場合では crush が成立しないので解析の必要がない。残った 29 通りの初期拍車列について考える。

プログラム公理から

$$\begin{aligned} L_0 \ \supset\ & \alpha\alpha = \alpha; 125/v \\ & \wedge\ \alpha\alpha\alpha = \alpha\alpha; 100/v, \\ R_0 \ \supset\ & \wedge\ \beta\beta \geq \beta; 2 \\ & \wedge\ \beta\beta\beta = \beta\beta; 6 \end{aligned}$$

が得られ、各拍車の立ち上がりの間隔が

$$\begin{aligned} \Theta \ \supset\ & \alpha\alpha = \alpha; 125/v \\ & \wedge\ \alpha\alpha\alpha = \alpha\alpha; 100/v \\ & \wedge\ \beta\beta \geq \beta; 2 \\ & \wedge\ \beta\beta\beta = \beta\beta; 6 \end{aligned} \tag{4.37}$$

でなければならない。たとえば図 4.4 は、初期拍車列 $(\alpha, \beta, \alpha, \beta, \beta, \alpha)$ の場合であり、危険領域中で 2 台の車が交差するため crush が成立する。拍車は $\alpha, \beta, \alpha, \beta, \beta, \alpha$ の順で駆動する訳だが、3 つめ (α の 2 回目) の駆動は時刻 $125/v$、4 つめ (β の 2 回目) の駆動は時刻 2、5 つめ (β の 3 回目) の駆動は時刻 6、6 つめ (α の 3 回目) の駆動は時刻 $100/v$ でそれぞれ行われる。これらの条件を満たすためには $100/v > 6$ かつ $125/v + 100/v > 2 + 6$ が成立しなければならず (図 4.4)、すなわち初期拍車列 $\langle\alpha\beta\alpha\beta\beta\alpha\rangle$ は $v < 16.6$ が成立する場合に限りありえる。$v \geq 16.6$ の場合には、プログラム公理からこの初期拍車列の順で拍車が駆動することはない。このようにして、残った 29 通りの初期拍車列うち、(4.37) を満たす初期拍車列は 15 通りである。他の 14 通りは v が仮定に反し、すなわち数学的にあり得ない場合である。最後に、*go* を満たさない場合を除外すると、4 通りが残った。すなわち、これらの初期拍車列は、プログラム公理から導かれた拍車の時間についての性質を満たし、*go* が成立するにもかかわらず *crush* を満たす、プログラム公理通りに動作して衝突が発生する場合を表す。

$$\begin{array}{lll} \langle\beta\alpha\beta\alpha\alpha\beta\rangle & \text{and} & 28.125 < v, \\ \langle\beta\alpha\beta\alpha\{\alpha\beta\}\rangle & \text{and} & 28.125 < v, \\ \langle\{\alpha\beta\}\beta\alpha\alpha\beta\rangle & \text{and} & 28.125 < v, \\ \langle\{\alpha\beta\}\beta\alpha\{\alpha\beta\}\rangle & \text{and} & 28.125 = v. \end{array}$$

直観的には、car_1 が交差点に到達したときに car_0 はまだ十分離れていたためアクセルを踏んだ場合、直後に相手の車が猛スピード (時速約 100km) でやって来た時にのみ衝突が起こる。

実時間問題の検証において、対象のプログラム系が、単純だが動作の場合分けが多いときにはこのように可能な初期拍車列を機械的に生成して数学的に条件に合わないものを除外していく方法が有効である。

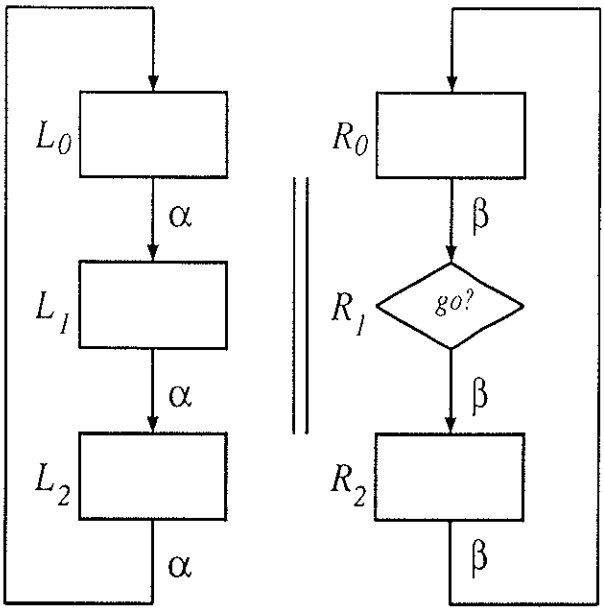


図 4.3: T 字路での合流問題のプログラム化

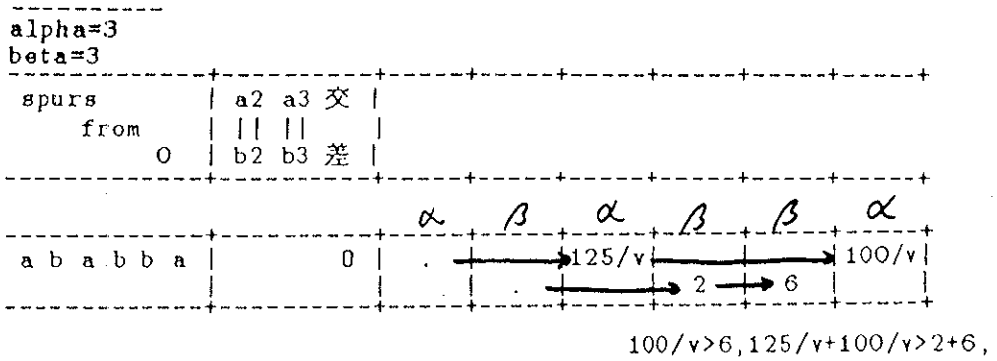


図 4.4: 初期拍車列 $\langle \alpha, \beta, \alpha, \beta, \beta, \alpha \rangle$

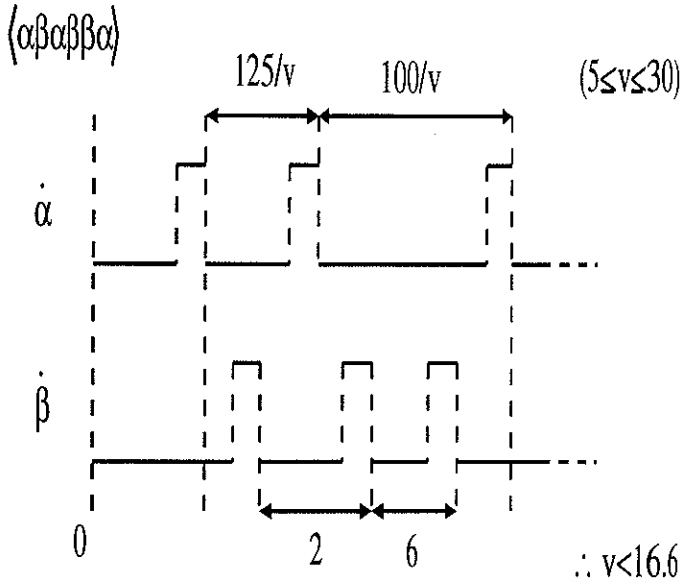


図 4.5: 拍車列と時間

表 4.4 T 字路での合流問題の解析

```
----------
alpha=3
beta=3
```

spurs from 0	a2 a3 交 || || b2 b3 差	.	125/v	100/v	.	2	6	
a a a b b b		.	125/v	100/v	.	2	6	
a a b a b b		.	125/v	100/v	.	2	6	
a a b b a b		.	125/v	100/v	.	2	6	100/v>2,125/v+100/v>2,
a a b b b a	0	.	125/v	100/v	.	2	6	100/v>2+6,125/v+100/v>2+6,
a a b b{ab}	0 0	.	125/v	100/v	.	2	6	100/v>2+6,
a a b{ab} b		.	125/v	100/v	.	2	6	100/v>2,
a a{ab} b b		.	125/v	100/v	.	2	6	
a b a a b b		.	125/v	100/v	.	2	6	100/v<2,
a b a b a b		.	125/v	100/v	.	2	6	100/v<2+6,125/v+100/v>2,
a b a b b a	0	.	125/v	100/v	.	2	6	100/v>6,125/v+100/v>2+6,
a b a b{ab}	0 0	.	125/v	100/v	.	2	6	100/v<2+6,125/v+100/v>2+6,
a b a{ab} b		.	125/v	100/v	.	2	6	100/v<2,125/v+100/v>2,
a b b a a b	0	.	125/v	100/v	.	2	6	100/v<6,125/v>2,
a b b a b a		.	125/v	100/v	.	2	6	125/v+100/v>2+6,125/v>2,
a b b a{ab}	0	.	125/v	100/v	.	2	6	125/v>2,125/v+100/v>2+6,
a b b b a a		.	125/v	100/v	.	2	6	125/v>2+6,
a b b{ab} a		.	125/v	100/v	.	2	6	125/v>2+6,
a b{ab} a b	0 0	.	125/v	100/v	.	2	6	100/v<6,125/v>2,
a b{ab} b a	0	.	125/v	100/v	.	2	6	100/v>6,125/v>2,
a b{ab}{ab}	0 0	.	125/v	100/v	.	2	6	100/v=6,125/v>2,
a{ab} a b b		.	125/v	100/v	.	2	6	100/v<2,
a{ab} b a b		.	125/v	100/v	.	2	6	100/v<2+6,100/v>2,

a{ab} b b a		0	.	125/v			100/v	XXXXX	100/v>2+6,
			.	2	6			XXXXX	
a{ab} b{ab}	0	0	.	125/v			100/v	XXXXX	100/v=2+6,
			.	2	6		XXXXX	XXXXX	
a{ab}{ab} b			.	125/v	100/v		XXXXX	XXXXX	100/v=2,
			.	2	6		XXXXX	XXXXX	
b a a a b b			.	125/v	100/v				125/v+100/v<2,
		.				2	6		
b a a b a b			.	125/v			100/v		125/v+100/v<2+6,125/v<2,
		.			2			6	
b a a b b a		0	.	125/v				100/v	100/v>6,125/v<2,
		.			2	6			
b a a b{ab}	0	0	.	125/v			100/v	XXXXX	125/v<2,125/v+100/v<2+6,
		.			2		6	XXXXX	
b a a{ab} b			.	125/v	100/v			XXXXX	125/v+100/v<2,
		.			2		6	XXXXX	
b a b a a b		0	.		125/v	100/v			100/v<6,125/v+100/v<2+6,
		.		2				6	
b a b a b a			.		125/v			100/v	125/v+100/v>6,125/v<2+6,
		.		2			6		
b a b a{ab}	0		.		125/v	100/v		XXXXX	125/v+100/v>6,125/v+100/v<2+6,
		.		2			6	XXXXX	
b a b b a a			.				125/v	100/v	125/v>6,
		.		2	6				
b a b{ab} a			.		125/v	100/v		XXXXX	125/v>6,125/v<2+6,
		.		2	6			XXXXX	
b a{ab} a b	0	0	.	125/v		100/v		XXXXX	100/v<6,125/v<2,
		.		2			6	XXXXX	
b a{ab} b a	0		.	125/v			100/v	XXXXX	100/v>6,125/v<2,
		.		2	6			XXXXX	
b a{ab}{ab}	0	0	.	125/v	100/v		XXXXX	XXXXX	100/v=6,125/v<2,
		.		2	6		XXXXX	XXXXX	
b b a a a b		0			.	125/v	100/v		125/v+100/v<6,125/v+100/v<2+6,
		.	2					6	
b b a a b a					.	125/v		100/v	125/v<6,125/v<2+6,
		.	2				6		
b b a a{ab}	0				.	125/v	100/v	XXXXX	125/v+100/v<6,
		.	2				6	XXXXX	
b b a b a a					.			125/v	100/v
		.	2			6			
b b a{ab} a					.	125/v	100/v	XXXXX	125/v<6,
		.	2			6		XXXXX	
b b b a a a							.	125/v	100/v
		.	2		6				
b b{ab} a a					.	125/v	100/v	XXXXX	
		.	2		6			XXXXX	
b{ab} a a b		0		.	125/v	100/v		XXXXX	125/v+100/v<6,
		.	2				6	XXXXX	
b{ab} a b a				.	125/v		100/v	XXXXX	125/v+100/v>6,125/v<6,
		.	2			6		XXXXX	

```
 b{ab} a{ab} |     0     |     |   .   |125/v|100/v|XXXXX|XXXXX| 125/v+100/v=6,
             |           |  .  |   2   |     |   6 |XXXXX|XXXXX|
-------------+-----------+-----+-------+-----+-----+-----+-----+
 b{ab} b a a |           |     |   .   |     |125/v|100/v|XXXXX| 125/v>6,
             |           |  .  |   2   |  6  |     |     |XXXXX|
-------------+-----------+-----+-------+-----+-----+-----+-----+
 b{ab}{ab} a |           |     |   .   |125/v|100/v|XXXXX|XXXXX| 125/v=6,
             |           |  .  |   2   |  6  |     |XXXXX|XXXXX|
-------------+-----------+-----+-------+-----+-----+-----+-----+
{ab} a a b b |           |  .  |125/v|100/v|     |     |XXXXX| 125/v+100/v<2,
             |           |  .  |     |     |  2  |  6  |XXXXX|
-------------+-----------+-----+-------+-----+-----+-----+-----+
{ab} a b a b |           |  .  |125/v|     |100/v|     |XXXXX| 125/v+100/v<2+6,125/v+100/v>2,125/v<2,
             |           |  .  |     |  2  |     |  6  |XXXXX|
-------------+-----------+-----+-------+-----+-----+-----+-----+
{ab} a b b a |        0  |  .  |125/v|     |     |100/v|XXXXX| 100/v>6,125/v+100/v>2+6,125/v<2,
             |           |  .  |     |  2  |  6  |     |XXXXX|
-------------+-----------+-----+-------+-----+-----+-----+-----+
{ab} a b{ab} |     0  0  |  .  |125/v|     |100/v|XXXXX|XXXXX| 125/v+100/v=2+6,125/v<2,100/v>6,
             |           |  .  |     |  2  |  6  |XXXXX|XXXXX|
-------------+-----------+-----+-------+-----+-----+-----+-----+
{ab} a{ab} b |           |  .  |125/v|100/v|     |XXXXX|XXXXX| 125/v+100/v=2,
             |           |  .  |     |  2  |  6  |XXXXX|XXXXX|
-------------+-----------+-----+-------+-----+-----+-----+-----+
{ab} b a a b |        0  |  .  |     |125/v|100/v|     |XXXXX| 100/v<6,125/v+100/v<2+6,125/v>2,
             |           |  .  |  2  |     |     |  6  |XXXXX|
-------------+-----------+-----+-------+-----+-----+-----+-----+
{ab} b a b a |           |  .  |     |125/v|     |100/v|XXXXX| 125/v+100/v>2+6,125/v<2+6,125/v>2,
             |           |  .  |  2  |     |  6  |     |XXXXX|
-------------+-----------+-----+-------+-----+-----+-----+-----+
{ab} b a{ab} |     0     |  .  |     |125/v|100/v|XXXXX|XXXXX| 125/v+100/v=2+6,125/v>2,100/v<6,
             |           |  .  |  2  |     |  6  |XXXXX|XXXXX|
-------------+-----------+-----+-------+-----+-----+-----+-----+
{ab} b b a a |           |  .  |     |     |125/v|100/v|XXXXX| 125/v>2+6,
             |           |  .  |  2  |  6  |     |     |XXXXX|
-------------+-----------+-----+-------+-----+-----+-----+-----+
{ab} b{ab} a |           |  .  |     |125/v|100/v|XXXXX|XXXXX| 125/v=2+6,
             |           |  .  |  2  |  6  |     |XXXXX|XXXXX|
-------------+-----------+-----+-------+-----+-----+-----+-----+
{ab}{ab} a b |  0     0  |  .  |125/v|100/v|     |XXXXX|XXXXX| 100/v<6,125/v=2,
             |           |  .  |  2  |     |  6  |XXXXX|XXXXX|
-------------+-----------+-----+-------+-----+-----+-----+-----+
{ab}{ab} b a |  0        |  .  |125/v|     |100/v|XXXXX|XXXXX| 100/v>6,125/v=2,
             |           |  .  |  2  |  6  |     |XXXXX|XXXXX|
-------------+-----------+-----+-------+-----+-----+-----+-----+
{ab}{ab}{ab} |  0  0     |  .  |125/v|100/v|XXXXX|XXXXX|XXXXX| 100/v=6,125/v=2,
             |           |  .  |  2  |  6  |XXXXX|XXXXX|XXXXX|
-------------+-----------+-----+-------+-----+-----+-----+-----+
total =63
----------
```

4.3 自動伴奏システム

CSP[6] を拡張したプログラム言語 CMP(Communicating Musical Processes) を提示し、制御信号から 0.5 秒の遅れを伴う MIDI 楽器 (グランドピアノ) を用いた自動伴奏システムの演奏速度を制御するプログラムをそれによって記述して、正当性を証明した [33]。

計算機による音楽研究は、自動演奏、楽曲及び奏法の分析、自動作曲、音響合成、楽譜読み取りなど、多種多様な分野にわたって様々なものが行われてきた。特に自動演奏は、ここ数年でハードウェア及びソフトウェア共に急速な発展を遂げ、楽譜の情報があれば誰でも自動演奏を楽しめるようになった。しかし、この演奏は楽譜に忠実な演奏であるため、単調で機械的なものとなり、人間が演奏する場合のような表情豊かな演奏にはならない。表情づけの試みの一つとして、自動演奏時に人間が実時間で指示することにより表情付を行なうシステム、自動演奏の実時間で制御システム ([17],[5] 他) が提案された。本研究では実時間制御システムの一形態、人間が演奏する主旋律に合わせての伴奏を自動的に行なう自動伴奏システムについて論じる。

自動伴奏システムにおいて、主旋律と伴奏部を各々並行プログラムとして捉えれば、正常な演奏を適当な範囲 (楽譜上の小節) でそれらが同期している状態であり CSP のランデブー概念を用いて記述できる。計算機が担当する伴奏部は、主旋律を演奏する人間の演奏速度を予測する関数 "est-tempo" がある範囲 (各命令の実行時間によって制限される) で正確ならばこのプログラムは正常に動作し、正常な演奏が行なわれることを証明した。

我々の研究グループでは入力信号に対し 0.5 秒遅れて反応する MIDI 楽器 (グランドピアノ) を制御して、人間の主旋律の演奏に合わせて伴奏する自動伴奏システムを実装中である。演奏の速さに注目すると、正常な演奏は主旋律と伴奏が適当な間隔で同期している状態と考えられる。CMP において、主旋律と伴奏を各々 1 つのプロセスとして捉え、この状態をランデブー概念を用いて記述した。さらにこれを Luckham が提唱している virtual text すなわち新たなプログラムによる仕様記述として捉え、システムの正当性を証明した。

4.3.1 virtual text

プログラムを作成するときは一般に注釈をつけてテキストの表面には現れないインフォーマルだが本質的な情報を付加し、人間がプログラムを読むときの助けとする。Luckham は [19] で注釈の概念を厳密化・形式化しそのプログラムがどの様に振舞うかを表す新たなプログラムを注釈に書くことを提唱している。こうして書かれた注釈のプログラムを virtual text と呼ぶ。virtual text は、「実行可能だが、実際には実行されないプログラム」であり、もとのプログラムの仕様を記述するもので実行速度やメモリに対する効率よりは人間にとって理解し易いことが重視される。そのために元のプログラムに比べより論理的な、あるいはより高級なプログラミング言語で記述されるべきである。

ここでは virtual text の記述言語として CMP を提示する。

4.3.2 CMP

CSP は入出力命令を基礎としたメッセージ通信型の並行計算モデルである。プログラムは静的に宣言された並行プロセスによって構成され、入出力命令は外部環境との情報のやりとり以外に、並行プロセス間の通信にも用いられる。入出力命令は、通信を行なう相手を指定し、バッファを経由せずに直接同期をとりながら相手プロセスとメッセージの送受を行なう。これがラン

デブーである。従って、プロセス内の入力(出力)命令は、相手プロセスの対応する出力(入力)命令が現れた時点で実行される。

CSP は計算モデルであるため、実用的なプログラミング言語と比較すると単純な基本命令しか用意されていないが、サブルーチンや再帰的手続きやセマフォ等を容易に実現できる。ここでは、CSP にリスト構造と、呼出し時の時刻を得る関数 **time**、与えられた音価の音を (MIDI ピアノで) 出力する命令 **play** を導入した CMP を用いる。

CMP の構文を以下に示す。ただし、$n \in \mathcal{N}$、$n \geq 1$、$c, c_1, \ldots, c_n \in \mathrm{CHAN}$、$x, x_1, \ldots, x_n \in \mathcal{V}_{\mathcal{Q}}$、$y \in \mathcal{V}_{\mathcal{L}}$、$\theta \in \mathcal{Q}$とする。また、$l_1, l_2$ のような添字付きの表現で l の形のものを表す。m, b, S についても同様である。

Arithmetic Expression	e ::=	$\theta \mid x \mid \mathbf{time} \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 \times e_2$
List	l ::=	$\mathbf{nil} \mid (\theta \ . \ l) \mid (l_1 \ . \ l_2)$
List Expression	m ::=	$l \mid y \mid \mathbf{first}(m) \mid \mathbf{rest}(m)$
Boolean Expression	b ::=	$e_1 = e_2 \mid e_1 < e_2 \mid m_1 = m_2 \mid \neg b \mid b_1 \vee b_2$
Statement	S ::=	$x := e \mid y := m \mid \mathbf{delay}\ e \mid \mathbf{play}\ n \mid c!e \mid c?x$
		$\mid S_1; S_2 \mid G \mid *G \mid S_1 \| S_2$
Guarded Command	G ::=	$[\Box_{i=1}^{n} b_i \rightarrow S_i] \mid [\Box_{i=1}^{n} b_i;\ c_i?x_i \rightarrow S_i \ \Box\ b; \mathbf{delay}\ e \rightarrow S]$

なお、$\wedge, \supset, \neq$ 等も通常の略記法に従い導入する。リストについては適当に省略して表記する。

guarded command について、$\Box$ で区切られた各部分の $\rightarrow$ の左側をガード、右側を本体という。

構文に関して次のような制約をもうける。

1. $S_1; S_2$ において、チャンネル c に対し
   (a) S_1 に c を用いる送信命令がある場合、S_2 に c を用いる受信命令があってはならない。
   (b) S_1 に c を用いる受信命令がある場合、S_2 に c を用いる送信命令があってはならない。
2. guarded command G に、$\Box$ で区切られた各部分間（それ自身を含む）の通信は許されない。
3. 並列に実行するプロセス間には共有変数がない。

4.3.3 解釈について

方向付きチャンネルの集合 DCHAN を次のように定義する。

$$\mathrm{DCHAN} = \mathrm{CHAN} \cup \{c! \mid c \in \mathrm{CHAN}\} \cup \{c? \mid c \in \mathrm{CHAN}\}$$

また、ステートをプログラム変数への値の割当てとする。すなわち、アトムが有理数か **nil** であるようなリスト全体の集合を $\mathcal{L}$ とすると、ステート s は $x \in \mathcal{V}_{\mathcal{Q}}$ のとき $s(x) \in \mathcal{Q}$、$y \in \mathcal{V}_{\mathcal{L}}$ のとき $s(y) \in \mathcal{L}$ とする。ステート全体の集合を STATE とする。

定義 16 (communication function) 時刻からチャンネルの状態への関数 *communication function* の集合 CF を次のように定義する。

$$CF = \{cf \mid cf : [0, t_0) \rightarrow \mathcal{P}((DCHAN - CHAN) \cup (CHAN \times \mathcal{Q})) \quad , \quad t_0 \in \mathcal{Q}^{\geq 0} \cup \{\infty\}\ \}$$

また、*communication function* cf の定義域が $[0, t_0)$ のとき、cf の期間 $|cf|$ を t_0 とする。

TA においてこれらは T^{Ω} から T^{Ω}_{C} に追加された定数としてすなわちプログラム変数を表す定数として扱う。

例えば

$$cf(5) = \{c_1!,\ c_2?,\ \langle c_3,\ 7\rangle\}$$

は、時刻 5 でチャンネル c_1 は送信の待ち、c_2 は受信の待ち、c_3 は実際に通信中で値 7 が受け渡されていることを表す。

モデルは (init, comm, final) の三つ組とする。ただし、init, final $\in$ STATE、comm $\in$ CF とする。

プログラムは、その実行が満たすようなモデル全ての集合として解釈される。

1. **delay** e は実行に e 秒 (e が負のときは 0 秒) かかり、その実行はステートに影響を及ぼさない。

2. **time** は呼出し時の時刻を得る関数とするが、どのプロセスからでも同時に呼び出せば同じ値が得られるものとする。

3. $c!e$ は、チャンネル c を通して値 e を送信する (ために待ち始める) 命令で、他のプロセスで同じチャンネル c についての受信 (を待ち始める) 命令 $c?x$ が実行されるまで待ち続ける。$c?x$ が既に実行されて待っていた状態ならば $c!e$ は待つ事なく通信が行なわれ、値 e が受け渡された後に双方の命令の実行が終了する。受信側では受け取った値を変数 x に格納する。

4. guarded command のうち、ガードに受信や **delay** 命令を含まないものは

   (a) ガードを評価して、それが 1 つも成立しない場合は終了する。

   (b) 成立するガードに対応する本体の 1 つが非決定的に選ばれ、実行される。

   ガードに **delay** 命令を含むものは、**delay** で指定された時間まで (その時間丁度は含まない) に他のガードが 1 つも成立しない場合にこれが選択実行される点が異なる。なお、受信命令を含むガードは、他のプロセスで対応する送信命令が実行されてその通信が実際に成立した場合に、成立するものとする。

5. guarded command G に対し $*G$ は iteration と呼ばれ、ガードが 1 つも成立しなくなるまで G を繰り返し実行する。

6. $S_1||S_2$ は S_1, S_2 のどちらかの実行が先に終了した場合、$S_1||S_2$ 全体としては他方が終わった時点で終了する。

モデル化に際して、代入、通信、ガードの評価、**play** 命令にかかる時間は各々一定と仮定し K_a, K_c, K_g, K_p で表す。これらは演奏する音楽の小節の長さに比べ十分に小さい値と仮定できる。TA においてこれらは T^{Ω} の定数としてすなわち本来の定数として扱う。

4.3.4 モデル化

正常な演奏を、主旋律と伴奏が各小節の最初について適当な範囲内で同期している状態とする。主旋律、伴奏を各々 1 つのプロセスと考えれば各小節の最初で

1. 伴奏が主旋律よりも δ 秒遅い

2. または主旋律が伴奏よりも δ' 秒遅い

場合に演奏をやめるプログラムを考えると、その主要な部分は次のように表せる。

$$
\begin{array}{rll}
\text{主旋律} & == & i := 1; \mathrm{cont} := 1; \\
 & & *[\mathrm{cont} = 1 \rightarrow \ i \text{ 小節目の主旋律の演奏} \\
 & & \quad \| \ [e!0 \rightarrow i := i + 1 \\
 & & \quad \Box \mathbf{delay}\ \delta \rightarrow \mathrm{cont} := 0] \ ] \\
\text{伴奏} & == & j := 1; \mathrm{cont}' := 1; \\
 & & *[\mathrm{cont}' = 1 \rightarrow \ j \text{ 小節目の伴奏の演奏} \\
 & & \quad \| \ [e?k' \rightarrow j := j + 1 \\
 & & \quad \Box \mathbf{delay}\ \delta' \rightarrow \mathrm{cont}' := 0] \ ]
\end{array}
$$

自動伴奏システムでは主旋律を人間、伴奏を計算機が担当する。簡単のために主旋律伴奏部共に単旋律である曲の演奏を考え、また、検証を容易にするために人間の演奏もプロセスとして表現する。すなわち人間の演奏データをあらかじめ持っていてそれを再現するプロセスを考える。そのために関数 perform を主旋律の小節内の各音をどのような間隔で鳴らすかを得るものとする。

$$\text{perform}: \text{小節番号} \mapsto (\ (\text{実演奏の音長}, \text{音高}), \cdots)$$

また、伴奏部の楽譜データとして関数musicを用意しておく。

$$\text{music}: \text{小節番号} \mapsto (\ (\text{正規化した音価}, \text{音高}), \cdots)$$

ただし、小節番号と音高は自然数、実演奏の音長は非負有理数（単位は秒）とする。正規化した音価とはその小節の拍数の合計を 1 としたときのその音の拍数とし、有理数で表される。

伴奏部のプロセスの動作は以下の 3 つの点を満たすべきであることに注意しなければならない。

1. 適切なタイミングで小節内の各音符を鳴らすために、人間の演奏速度を予測しなければならない。

2. MIDI 信号を受け取ってから 0.5 秒遅れて音を出すピアノを制御するために、各音符の適切なタイミングの 0.5 秒前に信号を送らなければならない。

3. ある小節の演奏中でも人間が次の小節を演奏し始めたなら、なるべくそれに合わせるべきである。

1 については、自然な演奏には流れがあることから、曲の中でのその小節の位置と、それまでの小節の実演奏からその小節の長さをある程度予測することが考えられる。そのような関数 est-tempo を仮定する。est-tempoは小節番号に多くの情報があるものとみなし、また模範的な演奏データも持っているものと考えて、現在の小節番号とそれまでの各小節の実演奏の長さの逆順リストから現在の小節の長さを予測するものとする。理想的なest-tempoがあれば上記の 1 は解決され、3 の前提は成立しない。perform, music, est-tempoは演奏する曲に依存する関数である。

4.3.5 自動伴奏プログラムの virtual text

4.3.4節で述べた注意点 2 と 3 を解決し、小節内の演奏をするプロセスを加えたプログラムを以下に示す。演奏はプログラム開始の 0.5 秒後から始めるものとし、第 1 小節の長さは est-tempo(1, nil) で適当な値が得られるものとする。演奏が終了するのは、主旋律と伴奏がある範囲内で同期しなかった場合とし、曲の終了の処理については省略する。

performは人間の演奏データである。このシステムは並行プロセス「人間」「人演」「指揮」「指演」「指ピ」から構成される。

自動伴奏プログラム == { 人間 || 人演 || 指揮 || 指演 || 指ピ }, ただし

$$
\begin{array}{lll}
\text{人間} == & i := 1; \text{cont} := 1; & \textbf{delay } 0.5; \\
& *[\text{cont} = 1 \rightarrow & a!i; t := \textbf{time}; \\
& & [e!t \rightarrow i := i + 1 \\
& & \ \square \textbf{delay } \delta \rightarrow \text{cont} := 0]; \\
& & a'?h]
\end{array}
$$

$$
\begin{array}{lll}
\text{人演} == & *[a?i' \rightarrow & x := \text{perform}(i'); \\
& & *[x \neq \textbf{nil} \rightarrow \textbf{play first}(\textbf{rest}(\textbf{first}(x))); \\
& & \quad \textbf{delay first}(\textbf{first}(x)); x := \textbf{rest}(x)\ ]; \\
& & a'!0]
\end{array}
$$

$$
\begin{array}{lll}
\text{指揮} == & t_0 := \textbf{time}; & \text{cont}' := 1; j := 1; t' := 0; t'' := 0; z := \textbf{nil}\ ; \\
& *[\text{cont}' = 1 \rightarrow & \textbf{delay } (\text{est-tempo}(j,\ z) - 0.5 - (t'' - t')); \\
& & d!j;\ d!\text{tempo}; t'' := \textbf{time} + 0.5; \\
& & [e?t' \rightarrow \ \text{tempo} := t' - t_0; t_0 := t'; \\
& & \quad j := j + 1; z := (\text{tempo}\ .\ z) \\
& & \square \textbf{delay} \quad (\delta' + 0.5) \rightarrow \text{cont}' := 0]\ ]
\end{array}
$$

$$
\begin{array}{lll}
\text{指演} == & y := \textbf{nil}\ ; z' := \textbf{nil}\ ; & \\
& *[d?j' \rightarrow & d?\text{tempo}'; y := \text{music}(j'); u := \textbf{first}(y); z' := (\text{tempo}'\ .\ z') \\
& \square y \neq & \textbf{nil}\ ; \textbf{delay } (\text{est-tempo}(j',\ z') \times \textbf{first}(u)) \\
& & \rightarrow u := \textbf{first}(y);\ c!\textbf{first}(\textbf{rest}(u)); y := \textbf{rest}(y);]
\end{array}
$$

$$
\begin{array}{lll}
\text{指ピ} == & k := 1; *[c?n \rightarrow & [k = 1 \rightarrow c_1!n; k := 2 \\
& & \square k = 2 \rightarrow c_2!n; k := 3 \\
& & \quad \cdots \\
& & \square k = 128 \rightarrow c_{128}!n; k := 1]] \\
& || * [c_1?n_1 \rightarrow \textbf{delay} & 0.5; \textbf{play } (n_1)] \\
& || * [c_2?n_2 \rightarrow \textbf{delay} & 0.5; \textbf{play } (n_2)] \\
& \quad \cdots & \\
& || * [c_{128}?n_{128} \rightarrow \textbf{delay} & 0.5; \textbf{play } (n_{128})]
\end{array}
$$

プロセス「人間」「指揮」がそれぞれ主旋律、伴奏にあたる。実際にはプロセス「人間」と「人演」を人間が、「指ピ」を MIDI ピアノが担当する。「人演」「指演」等の **delay** は正確には K_a, K_c, K_g, K_p を考慮したものでなければならないが、これらの値は十分に小さいので省略する。

4.3.6 正当性とest-tempo

4.3.5で示した自動伴奏プログラムのモデル $\sigma = \langle \sigma_{\text{init}},\ \sigma_{\text{comm}},\ \sigma_{\text{final}} \rangle$ について考える。

チャンネル c、 arithmetic expression e について通信 $\langle c,\ e \rangle$ の始まる時刻 $\uparrow \langle c,\ e \rangle$ を以下のように定める。

$$\uparrow \langle c,\ e \rangle \stackrel{d}{=} \inf\{t |\ \langle c,\ e \rangle \in \sigma_{\text{comm}}(t)\}$$

任意の小節番号 i について、 $\eta(i)$ をその小節の主旋律の音符の数とし、人間の演奏データが次のように表されるとする。

$$\text{perform}(i) = (\ (\text{長}_{i,1},\ \text{音}_{i,1})\ ,\ \cdots\ ,\ (\text{長}_{i,\eta(i)},\ \text{音}_{i,\eta(i)})\ )$$

ここで任意の小節番号 i について、

$$A_i \stackrel{\text{def}}{\equiv} \ -\delta - \Delta_i < T_i^{play} - T_i^{est} < \delta' + \Delta_i$$

とおく。ただし、 $\eta(i)$ をその小節の主旋律の音符の数とし、

$$\Delta_i \stackrel{\text{def}}{=} \ -2 \times K_c + (\eta(i) - 2) \times K_a + \eta(i) \times K_g + \eta(i) \times K_p$$

$$T_i^{play} \stackrel{\text{def}}{\equiv} \text{ i 小節目の人間の演奏の時間}$$

$$T_i^{est} \stackrel{\text{def}}{\equiv} \text{ est-tempoによる i 小節目の演奏時間の予測}$$

とする。 Δ_i は i 小節の演奏時に主旋律で代入・通信等に費やされる時間から伴奏で代入・通信等に費やされる時間を引いたものであり、十分に小さな値である。 A_i は i 小節の人間の演奏の長さとest-tempoによって得られた予測が $\delta + \Delta_i,\ \delta' + \Delta_i$ の範囲で一致していることを意味する。

$$A_i \ \equiv \ -\delta - \Delta_i \ < \ \sum_{j=1}^{\eta(i)} \text{長}_{i,j} \ - \ \text{est-tempo}(i,\ z_i) \ < \ \delta' + \Delta_i$$

とする。ただし、

$$\Delta_i = -2 \times K_c + (\eta(i) - 2) \times K_a + \eta(i) \times K_g + \eta(i) \times K_p$$

$$z_i = \begin{cases} \textbf{nil} & \text{if } \ i = 1 \\ (\uparrow \langle a,\ i \rangle - \uparrow \langle a,\ i-1 \rangle\ .\ z_{i-1}) & \text{otherwise} \end{cases}$$

とする。 Δ_i は i 小節の演奏時に主旋律で代入・通信等に費やされる時間から伴奏で代入・通信等に費やされる時間を引いたものであり、十分に小さな値である。 A_i は i 小節の人間の演奏の長さとest-tempoによって得られた予測が $\delta + \Delta_i,\ \delta' + \Delta_i$ の範囲で一致していることを意味する。また、

$$t_i' = \uparrow \langle a,\ i \rangle + K_c,\ \ t_i'' = \uparrow \langle d,\ i \rangle + 2 \times K_c + 0.5$$

と定義する。

仮定より各小節の $-\delta \ < \ t_i'' - t_i' \ < \ \delta'$ ならば $i+1$ 小節も演奏される。逆も真である。ここで

$$-\delta - \Delta_i \ < \uparrow \langle a,\ i \rangle - (\uparrow \langle d,\ i \rangle + 0.5) \ < \ \delta' + \Delta_i$$

$$\supset \quad -\delta \ < \ t_i'' - t_i' \ < \ \delta'$$

であり、

$$A_i \quad \supset \quad -\delta - \Delta_i \ < \uparrow \langle a,\ i+1 \rangle - (\uparrow \langle d,\ i+1 \rangle + 0.5) \ < \ \delta' + \Delta_i$$

すなわち、A_i を満たすような perform, est-tempo ならば $i+1$ 小節の最初で主旋律と伴奏が同期する。

以上より、

$$\forall i \in \mathcal{N}^+ \ (\ A_i\ ) \qquad \cdots \quad (\alpha)$$

を満たすような perform, est-tempo ならば主旋律と伴奏が適当な範囲内で同期し、正常な演奏を続ける。ただし $\mathcal{N}^+$ は正整数全体の集合とする。

$$-\delta - 3K_a + K_g + 2K_c < \text{伴} - \text{主} < \delta' + 3K_a + K_g + 2K_c$$

TA でこの証明を行なう場合、自動伴奏システムの各プロセスおよびチャネルを special constant とし、プログラム公理として各時間値について定式化する。

人間の主旋律に計算機が伴奏を合わせる立場では、予想されるperformつまり人間の標準的な演奏について、式 (α) を満たすようなest-tempoを採用する必要がある。

これらを用い、次の定理が導ける [33]。ただし $\mathcal{N}^+$ は正整数全体の集合とする。

定理 9 条件

$$\forall i \in \mathcal{N}^+ \ (\ A_i\ ) \qquad \cdots \quad (A)$$

を満たすような *perform*, *est-tempo* ならば

$$T_i^{acc} = i\text{小節目の伴奏の開始時刻}$$

$$T_i^{princ} = i\text{小節目の主旋律の開始時刻}$$

とすると

$$-\delta - 3K_a - K_g - 2K_c < T_i^{acc} - T_i^{princ} < \delta' + 3K_a + K_g + 2K_c$$

が成立、すなわち主旋律と伴奏が適当な範囲内で同期し、正常な演奏を続ける。

人間の主旋律に計算機が伴奏を合わせる立場では、予想されるperformつまり人間の標準的な演奏について、式 (A) を満たすようなest-tempoを採用する必要がある。

第 5 章

結論および今後の展望

並行プログラム系の実時間における動作検証のための形式的体系 tense arithmetic を提示した。この体系により実際の時間値が現われる具体的なレベルのプログラム検証を、厳密に行なうことが可能である。また拍車により真に並行動作する実時間プログラムの定式化が容易となり、プログラムの動作の場合分けは拍車の列でインデックスづけて扱うことができる。

tense arithmetic による並行プログラム系の検証例としては、まず Dekker の解の飢餓回避問題を証明した。このとき実行時間の上限や下限が制約として伝播することから場合分けの枝刈りを行うことができ、有効であった。プログラムの 1 ステップごとの状態における推論から、目的の状態に到達するまでの遷移の過程および、それにかかる時間の上限と下限も判明した。次に、T 字路における合流問題では各ビークルの位置を簡単なプログラムとして定式化した後、主に機械的な方法を採り入れて検証を進めた (拍車の系列の生成、各系列がプログラム公理を満たすか否かのチェック)。この方法は将来、証明支援システムを構築する際の手法の一つとして有効である。結果として、与えられた条件のもとではビークルの衝突が発生してしまう場合があることを導いた。また、遅れを伴う MIDI 楽器を用いた自動伴奏システムの演奏速度を制御するプログラムの virtual text を、CSP を拡張したプログラム言語においてランデブー概念を用いて記述し、その正当性の証明を tense arithmetic で行なう指針を示した。

tense arithmetic は形式的体系の厳密さを失わずに、より実践的な応用を可能とした検証体系である。

命題の成立する時間に注目し、証明の最中には主に時間の値と拍車の駆動する順列を扱うためユーザーは論理的というよりむしろ数学的に証明を遂行すればよく、そこで有理数理論を利用できるため簡便に証明が可能でかつ厳密に論理的裏づけがされる。すなわち高度に論理的な手法と実地におけるシステム設計・検証との間の橋渡しを可能としている。

現在、TA は外部の連続系を直接制御するプログラムを扱うように体系が構築されていないが、拡張するには $T^{\mathbb{Q}}$ から $T^{\mathbb{Q}}_{C}$ に定数記号を追加するときにプログラム変数と同様に連続系を表すための連続変数 (*continuous variable*) 及びその導関数を special constant として追加し、それらの解釈を補い公理を修正・追加する。その場合、軌跡の公準は連続変数及びその導関数については適用せず連続的な変化を扱えるようにする。

今後は tense arithmetic を用いてさらに並列コンピュータやディスクアレイ装置、論理回路、交通システム等広範囲に渡って解析・検証をし、外部の連続系を含め必要に応じて記述力の強化や体系の整備を行ないたい。証明の自動化についても追求する。また 3 章でプログラム検証においては時相演算子のネスティングが必要な場面が限られることに触れたが、様々な検証の場面において真に必要とされる時相演算子のネスティングについて厳密な考察を行ないたい。

謝辞

この研究をすすめるにあたり多大な御指導をいただいた筑波大学教授五十嵐滋先生、千葉大学教授辻尚史先生、筑波大学助教授細野千春先生、筑波大学講師水谷哲也先生ならびに筑波大学助手塩雅之先生に心より感謝致します。また、予備審査以来有益な御助言をいただいた筑波大学教授井田哲夫先生、筑波大学教授坂本直人先生、筑波大学教授田中二郎先生、研究の議論に参加していただいた埼玉短期大学講師池田靖雄先生、工業技術院機械技術研究所富田康治さん、研究室のOBである畑中秀行さんに感謝いたします。最後になりましたが、いろいろな励ましをいただいた筑波大学人工知能研究室の皆様どうもありがとうございました。

参考文献

[1] Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T. A., Ho, P. -H., Nicollin, X., Olivero, A., Sifakis, J. and Yovine,S. : The Algorithmic Analysis of Hybrid Systems, *Theor. Comp. Sci., North-Holland Publ. Co., Amsterdam*, **138** (1995), pp. 3–34.

[2] Chaochen, Z. : Duration Calculi: an Overview, *International Institute for Software Technology, The United Nations University, UNI/IIST Report* **10**, 1993.

[3] Dijkstra, E. W. : Co-operating Sequential Processes, *Programming Languages* (1968), pp. 43–112.

[4] Gentzen, G. : Untersuchungen über das logische Schließen, *Mathematische Zeitschrift, Verlag von Julius Springer* **39** (1935), pp. 176–210, 450–431; Investigations into Logical Deduction, Szabo, M. E. (ed.), *The Collected Papers of Gerhard Gentzen, Series of Studies in Logic and the Foundations of Mathematics, North-Holland Publ. Co.* (1969), pp. 68–131.

[5] 原口 剛 : 自動演奏における実時間制御技術の研究, 筑波大学理工学研究科修士論文, 1994.

[6] Hoare, C. A. R. : *Communicating Sequential Processes*, Prentice-Hall Internat., London, 1985.

[7] Hooman, J. : *Specification and Compositional Verification of Real-time Systems*, Springer-Verlag, 1991.

[8] Igarashi, S. : The ν-Conversion and an Analytic Semantics, R. E. A. Mason (ed.), *Inf. Proc. 83, Elsevier Sci. Publ. B. V.* (1983), pp. 769–774.

[9] Igarashi, S., Mizutani, T. and Tsuji, T. : An analytical semantics of parallel program processes represented by ν-conversion, *Tensor, N. S.* **45** (1987), pp. 222–228.

[10] Igarashi, S., Tsuji, T., Mizutani, T. and Haraguchi, T. : Experiments on Computerized Piano Accompaniment, *Proc. of International Comput. Music Conf.* (1993), pp. 415–417.

[11] Igarashi, S., Mizutani, T., Tsuji, T. and Hosono, C. : On Locomorphism in Analytical Equivalence Theory, *Logic, Language and Computation, Lecture Notes in Computer Science* **792** (1994), pp. 173–187.

[12] Igarashi, S., Mizutani, T., Shirogane, T. and Shio, M. : Formal Analysis for Continuous Systems Controlled by Programs, *Concurrency and Parallelism, Programming, Networking, and Security, Lecture Notes in Comp. Sci., Springer-Verlag, Berlin*, **1179** (1996), pp. 347–348.

[13] Igarashi, S., Shio, M., Shirogane, T. and Mizutani, T. : *Formal Verification and Evaluation of Execution Time in the Envelope Theory*, ibid., (1996), pp. 299–308.

[14] 五十嵐滋, 水谷哲也, 塩雅之, 白銀哲也, 富田康治 : 知的制御プログラムの数理的基礎について, 第 38 回プログラミング・シンポジウム報告集, (1997), pp. 1 – 12.

[15] Igarashi, S., Shirogane, T., Shio, M. and Mizutani, T. : Tense Arithmetic I: Formalization of Properties of Programs in Rational Arithmetics, *Tensor, N. S.* **59** (1998), to appear.

[16] 北島 伸克 : 有理数時間上の $\kappa o \sigma \mu o \varsigma$ に基づくプログラム表現と検証, 筑波大学大学院博士課程工学研究科修士論文, 1992.

[17] 小宮山 弘樹 : 自動演奏システムの形式的記述とその表現, 同上, 1993.

[18] Kröger, F. : *Temporal Logic of Programs*, Springer-Verlag, 1987.

[19] Luckham, D. : *Programming with Specifications*, Springer-Verlag, 1990.

[20] Majumdar, R. and Shyamasundar, R. K. : Design of Controllers for Linear Hybrid Systems, *Concurrency and Parallelism, Programming, Networking, and Security, Lecture Notes in Comp. Sci., Springer-Verlag, Berlin*, **1179** (1996), pp. 309–320.

[21] Manna, Z. and Pnueli, A. : Completing the Temporal Picture, *Theor. Comp. Sci.*, **83** (1991), pp. 97-130.

[22] Manna, Z. and Pnueli, A. : *The Temporal Logic for Reactive and Concurrent Systems: Specification*, Springer-Verlag, 1992.

[23] 水谷哲也, 五十嵐滋, 小宮山弘樹, 辻尚史 : 一二の並行プロセスの検証問題について, 第 34 回プログラミングシンポジウム報告集 (1993), pp. 105–116.

[24] Mizutani, T., Igarashi, S., Tomita, K. and Shio, M. : Representation of Discretely Controlled Continuous Systems in Software-oriented Formal Analysis, *Advances in Computer Science, Lecture Notes in Comp. Sci. Springer-Verlag, Berlin*, **1345** (1997), pp. 110–120.

[25] Mizutani, T., Shio, M., Shirogane, T. and Igarashi, S. : Tense Arithmetic II: Representation and Verification of Discretely Controlled Systems, *Tensor, N. S.*, (submitted to).

[26] Moszkowski, B. C. and Manna, Z. : Reasoning in interval temporal logic, *Proceedings of the ACM/NSF/ONR Workshop on Logic of Programs, Lecture Notes in Comp. Sci. Springer-Verlag, Berlin*, **164** (1984), pp. 371–383.

[27] Moszkowski, B. C. : *Executing Temporal Logic Programs*, Cambridge Univ. press, 1986.

[28] Pnueli, A. : A. M. Turing Award Lecture: Verification Engineering: a Future Profession, *Proceedings of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing* (1997), p. 7.

[29] 塩 雅之, 五十嵐 滋, 辻 尚史, 水谷 哲也, 白銀 哲也 : 時間の論理の束モデルの 2 次元的解釈, 応用数学合同研究集会報告集 (1994), pp. 5-1 – 5-6.

[30] 塩 雅之 : エンヴェロープ理論とその応用, 筑波大学大学院博士課程工学研究科学位論文, 1998.

[31] 白銀哲也, 五十嵐滋, 辻尚史, 水谷哲也 : 時間の論理の束モデル, 1993 年度応用数学合同研究集会報告集 (1993), pp. 31 – 34.

[32] 白銀哲也, 五十嵐滋, 辻尚史, 細野千春, 水谷哲也 : 時間の論理の束モデルの拡張, 1994 年度応用数学合同研究集会報告集 (1994), pp. 6-1 – 6-3.

[33] 白銀哲也, 五十嵐滋, 辻尚史, 水谷哲也 : ランデブーを用いた自動伴奏システムのモデル化と検証, 1994 年度応用数学合同研究集会報告集 (1994), pp. 9-1 – 9-6.

[34] 白銀哲也, 五十嵐滋, 水谷哲也, 塩雅之 : 有理拍車に基づいたプログラムの検証ないし実時間動作解析, 1995 年度応用数学合同研究集会報告集 (1995), pp. 21-1 – 21-6.

[35] 白銀哲也, 五十嵐滋, 塩雅之, 水谷哲也 : 有理拍車を伴うシステムの表現と検証, 日本ソフトウェア科学会第 13 会大会論文集 (1996), pp. 21–24.

[36] 白銀哲也, 五十嵐滋, 塩雅之, 水谷哲也 : Tense Arithmetic を用いた並行プログラム系の解析と検証, 1998 年度応用数学合同研究集会報告集 (1998), pp. 71 – 76.

[37] Takeuti, G. : *Two Applications of Logic to Mathematics*, Princeton Univ. Press, Princeton, 1978.

[38] 富田康治, 辻尚史, 五十嵐滋 : プログラムにおける実時間問題の ν- 転換による解析と動作条件, 情報処理学会論文誌, **34** (1993), pp. 1099–1106.

[39] Tomita, K., Tsuji, T. and Igarashi, S. : Analysis of a Software/Hardware System by Tense Arithmetic, *Logic, Language and Computation, Lecture Notes in Computer Science*, **792** (1994), pp. 188–205.

[40] Xuandong, L. and Hung, D. V. : Checking Linear Duration Invariants by Linear Programming, *Concurrency and Parallelism, Programming, Networking, and Security*, ibid., **1179** (1996), pp. 321–330.

[41] Xuandong, L., Hung, D. V. and Tao, Z. : Checking Hybrid Automata for Linear Duration Invariants, *Advances in Computer Science*, ibid., **1345** (1997), pp. 166–180.

[42] 鈴木 康人, 米崎 直樹 : 普通の Until 演算子を持つ命題実時間論理について, 情報処理学会第 50 回 (平成 7 年前期) 全国大会講演論文集 (1995), pp. 4-307 – 4-308.

付録.　tense arithmetic で用いる LK の推論規則

推論規則の名前については元々のLKのものに従っている。上から cut の規則までが構造に関する推論規則である。元々のLKの体系は型無し、単一ソートであったが tense arithmetic は2ソート (tense term は有理数値をとるものと有理数値または無限大の値をとるものの二種類ある) であるため∀-IA と ∀-IS はそれぞれ二つずつ用いる。ここで s は tense term、e は $T^{\mathbb{Q}}$ の項、z (u) は下式には含まれない *eigenvariable* と呼ばれる量化記号の出現しない tense (rational) term である。

Thinning: $\dfrac{\Gamma \to \Theta}{F, \Gamma \to \Theta}$, $\dfrac{\Gamma \to \Theta}{\Gamma \to \Theta, F}$;

Contraction: $\dfrac{F, F, \Gamma \to \Theta}{F, \Gamma \to \Theta}$, $\dfrac{\Gamma \to \Theta, F, F}{\Gamma \to \Theta, F}$;

Interchange: $\dfrac{\Delta, F, G, \Gamma \to \Theta}{\Delta, G, F, \Gamma \to \Theta}$, $\dfrac{\Delta \to \Theta, F, G, \Lambda}{\Delta \to \Theta, G, F, \Lambda}$;

Cut: $\dfrac{\Gamma \to \Theta, F \quad F, \Delta \to \Lambda}{\Gamma, \Delta \to \Theta, \Lambda}$;

∨-IA: $\dfrac{F, \Gamma \to \Theta \quad G, \Gamma \to \Theta}{F \vee G, \Gamma \to \Theta}$;

∨-IS: $\dfrac{\Gamma \to \Theta, F}{\Gamma \to \Theta, F \vee G}$, $\dfrac{\Gamma \to \Theta, G}{\Gamma \to \Theta, F \vee G}$;

¬-IA: $\dfrac{\Gamma \to \Theta, F}{\neg F, \Gamma \to \Theta}$;

¬-IS: $\dfrac{F, \Gamma \to \Theta}{\Gamma \to \Theta, \neg F}$;

∀-IA: $\dfrac{F[s],\ \Gamma \to \Theta}{\forall x F[x],\ \Gamma \to \Theta}$, $\dfrac{F[e],\ \Gamma \to \Theta}{\forall r F[r],\ \Gamma \to \Theta}$;

∀-IS: $\dfrac{\Gamma \to \Theta,\ F[z]}{\Gamma \to \Theta,\ \forall x F[x]}$, $\dfrac{\Gamma \to \Theta,\ F[u]}{\Gamma \to \Theta,\ \forall r F[r]}$.

本学関係