修 士 論 文 概 要 書

CD

2008 年 2 月提出　　学籍番号 3606U079 – 7

専攻名	情報・ネットワーク	氏 名	西村　光弘	指導教員	上田　和紀　印
研究指導名	並列知識情報処理				
研究題目	ハイブリッド並行制約プログラミングにおける制約の階層化				

1 ハイブリッド並行制約プログラミング

ハイブリッド並行制約プログラミング言語 (以下 HCC) はハイブリッドシステム（離散的変化と連続的変化からなるシステム，例えば物体の移動と衝突のモデル）の表現に適した言語である．現実世界を表現する際に用いる常微分方程式を制約として扱う．その制約を処理系内部で高精度な求解アルゴリズムを用いて計算しているため，複雑な計算手法は不要となり解きたい問題の記述のみでサンプリングデータを生成することができる．

2 研究の背景と目的

現状の HCC 処理系では制約に強さ（優先度）の概念は導入されていない．その問題点は，どの制約も必ず満たされなければならず，ユーザは全制約間で１つも矛盾がなくコードを記述する必要がある．並行言語でもあるがゆえに処理の流れを追うことが困難であり，また，ユーザによる制約間のチェックは制約の数の組み合わせの個数でほぼ増大する．プログラミング言語でのバグを現実的な時間内に全ては解消しきれないとすると，HCC にもある程度の柔軟性は必要である．その解決策として制約階層が有効であると考える．

本研究の目的は，コーディングにおいて制約過多，制約不足といった状況の解消支援をするため，制約の優先度を可能にする制約階層を実現することである．関連研究に比較子と呼ばれる同階層上で最適解を導き出す研究や，制約階層で制約の強さを適切に利用することで、UI においてユーザ支援の研究が行われている．従来の対象は幾何制約であったものに対し本研究は時間概念のある非線形制約も含み，目的も異なるため適用可能性，有効性を示すことは意義があると考える．

3 制約階層

制約階層は，硬い制約と軟らかい制約という概念を，強さと呼ばれる，有限段階の優先度へ拡張したものである．いわゆる硬い制約が最も強く，必須制約 (required) と呼ばれ，それ以外の軟らかい制約は選好制約 (preferential) と呼ばれる．通常，必須制約の強さは required として表され，選好制約の強さは，強いものから順に strong, medium, weak として表される．

制約階層で制約の強さを適切に利用することで，UI の構築が容易になる．関連研究ではグラフィカルオブジェクトの振舞いの表現に有効であることを示すものがある．本研究では，プログラミング言語における UI は言語の記述性と考え，階層化が有効であるかを検証している．

4 HCC での制約階層の定義

4.1 syntax

ContConstr、Dconstr ついては論文参照

ac は ask 制約、N は数値、x は変数、X() は関数

$$HAgent :== Preferencial\ Agent \mid Agent$$

$$Preferencial :== "required" \mid "strong" \mid "medium" \mid "weak"$$

$$\begin{aligned} Agent :== & ContConstr \mid DConstr \mid \{Agent\} \\ & \mid if\ ac\ then\ Agent \mid if\ ac\ then\ Agent\ else\ Agent \\ & \mid unless\ ac\ then\ Agent \mid hence\ Agent \\ & \mid always\ Agent \mid do\ Agent\ watching\ ac \\ & \mid do\ Agent\ hencewatching\ ac \mid do\ Agent\ until\ ac \\ & \mid do\ Agent\ henceuntil\ ac \mid while\ ac\ do\ Agent \\ & \mid time\ Agent\ on\ ac \mid when\ ac\ do\ Agent \\ & \mid wait\ N\ do\ Agent \mid new\ x\ in\ Agent \\ & \mid X(t1,...,tn) \mid forall\ X(Y)\ do\ Agent \\ & \mid evalexpr\ Variable\ =\ Expression\ in\ Agent \\ & \mid sample(x1,...,xn) \end{aligned}$$

4.2 semantics

- c は制約
- Γ はエージェントの集合
- σ はストア
- $\sigma^\star$ は σ からある制約 c を除いた制約の集合
  c が σ に tell されている状態（入っている状態）は，c と $\sigma^\star$ とが矛盾しない
- $next, else$ は論文参照
- P（Preference）は優先度であり，優先度の高いものほど値が低い

現処理系の Tell

$$\langle(\Gamma, c), \sigma, next, else\rangle \rightarrow \langle\Gamma, (\sigma \cup \{c\}), next, else\rangle$$

$$\frac{(\sigma^\star \not\vdash \{c1 \wedge c2\})\ \wedge\ P.c1 < P.c2}{\langle(\Gamma, c1), (\sigma^\star, c2), next, else\rangle \rightarrow \langle\Gamma, (\sigma^\star \cup \{c1\}), next, else\rangle}$$

表 1: 追加した Tell2-1

$$\frac{(\sigma^\star \not\vdash \{c1 \wedge c2\}) \ \wedge \ P.c1 > P.c2}{\langle(\Gamma, c1), (\sigma^\star, c2), next, else\rangle \rightarrow \langle\Gamma, \sigma, next, else\rangle}$$

表 2: 追加した Tell2-2

5 階層化の有効性

5.1 フールプルーフ設計

制約の有無の観点からプログラミングを考えると，全時間において制約が無いという時間がなく，かつ矛盾する制約が２つ以上存在してはいけないというのが現状である．制約階層を用いると選好制約の weak で代表されるデフォルトとなる制約によって，全時間において制約が無いという時間は存在しなくなる．さらに，同階層の制約の整合性が保てるなら矛盾する制約がいくら存在してもかまわないことになる．

5.2 記述性の多様化・デバグへの有効性

現状のプログラミング手法の書き換えとして、if else の構文，状態遷移モデル，タグを使用するための unless 構文などがある．また，矛盾する制約を条件分岐を用いて解消する方法が主流であったが，制約階層を用いるとより明示的に解消することが可能となる．これはデバグにおいても有効である．なぜなら，現状は時間とともに移り変わる変数で判定する if 文がリダクションされているかを考慮にいれ，全制約間の矛盾のチェックをする必要があった．それを明示的に記述してある同階層間の制約のみをチェックするだけで良くなる．更に，実装の過程で矛盾する制約を見つける必要性があることが分かったので，矛盾する制約をユーザに提示できれば，ユーザは制約のチェックをする必要がなくなる．

5.3 可読性の向上

異なるクラス間では，階層構造をなしている if 文の中に用いられている if と else の内容を別々に記述することが難しい．そのため，片方のクラスで if 文を用いると，もう片方のクラスでタグをつけ unless 文を用いるのが主流となっている（主流となっている理由は省略する）．これも制約階層を用いることで、if 文の階層構造を小さくできるので、if A と if¬A を使うことが容易になる．つまり，c 言語でいう goto 文に近い意味合いのタグを多用することが減るのでコードの可読性が高まる．

6 実行処理アルゴリズム

書き換え規則アルゴリズム、HCC 処理系外部から操作による階層化アルゴリズムの実験と考察を踏まえて以下のアルゴリズムを設計した．

6.1 HCC 処理系内部のストア入出力操作による階層化アルゴリズム

他２つのアルゴリズムの問題点（フェーズごとに制約をチェックする．ポイントフェーズから値を引き継いで新たに始めることができないこと）を踏まえて、point phase と interval phase を別々にエラーが発生するかみる．point phase と interval phase の処理はほぼ同様なので１つのほうのアルゴリズムを示す．

Γ はキュー，σ は制約のストア，Σ は矛盾する制約のストア、A はエージェント、P は優先度 (値が低いほうが優先度が高い)、C は制約、ERR はエラー

1. parsing
2. dequeue(A) (from Γ)
3. store(P, A)
4. tell A
5. if ¬ ERR then compulating, and goto 2
   else goto (a)
   (a) which constrant($C2 \in \sigma$) does last executed Agent($C1 \in A$) contradict?
   (b) switch ($a = P.C1 - P.C2$){
   case $a > 0$:
   goto 2
   case $a < 0$:
   $\Sigma \cup \{C2\}$
   continue
   case $a = 0$:
   return ERR
   }
   (c) $initialize(\sigma)$, $initialize(\Gamma)$
   (d) parsing except for Σ
   (e) goto 2

7 まとめと今後の課題

ハイブリッド並行制約プログラミングに制約階層を適用することにより，記述性の多様化・デバグへの有効性・フールプルーフ設計・可読性の向上を評価基準に多くの例題をもとに検証を行い有効性を示した．特にフールプルーフ設計の点では，コーディングにおいて制約過多，制約不足といった状況の解消支援をする本研究の目的に適した検証が行えたと思われる．

また，３通りのアルゴリズムに従い，階層化設計について各々実験を行った．その実験結果をもとに現状最良と思われる設計を考案した．設計の中の１つの項目である矛盾しあう２つの制約を見つけるというアルゴリズムはそれだけでもデバグに非常に有効である．

今後の課題として２点あげる．１点目は，実装し，評価することで設計したアルゴリズムが実際に正しく挙動するかを証明することである．２点目は制約を時間ごとに強弱を入れ替えたりする操作を可能とする制約階層を実現することと，有限の階層化をユーザの指定によって半無限の階層化を実現できる仕様にすることである．これによって，より記述の多様性が増し，多くの制約に対し明示的に強弱が宣言できる．それに加え，コード量を激減させうるので，本当の意味で可読性が高まる．

参考文献

[1] 細部博史:ユーザインタフェースのための線形等式・不等式制約解消系　コンピュータソフトウェア,Vol.19,No.6,pp.13-20, 日本ソフトウェア科学会,2002.11.

[2] Alan Borning, Bjorn Feldman-Benson, Molly Wilson:Constraint Hierarchies,1992, OOPSLA '87 Proceedings, pages 48-60, October, Page 27 of 31

[3] Bjorn Carlson and Vineet Gupta：Hybrid cc with interval constraints,1997

[4] Bjorn Carlson and Vineet Gupta:The hcc Programmer's Manual,1999