Javaプログラム最適化の宣言的記述とその効率的な実装

番 伸宏[†]　胡 振江[†] [‡]　筧 一彦[†]　武市 正人[†]

概 要

本論文では Java 最適化フレームワーク Soot 上で記述された最適化処理作成器について述べる．このツールはユーザから与えられた最適化の仕様記述に基づきプログラムの書き換えを行なう．仕様記述は命令文の条件付書き換え規則を用い, CTL を拡張した時相論理で記述された条件を満たす時に, 命令文を書き換える．このツールを用いることより, 不要命令除去や定数伝播などの一般的なコンパイラ最適化技法の仕様を宣言的な記述で簡潔に表現でき, また, その処理を自動で高速に実行することができる．

1　はじめに

コンパイラの設計においては, コード最適化は重要なパスの一つであり, 命令を書き換え, 効率の良いコードを生成し, 実行速度を向上させたり, コードのサイズを小さくしたりする．

最適化のためのプログラム解析の仕様は, 従来は宣言的な手法でなく, 自然言語や疑似コードで記述されてきた．このような記述からは最適化処理を自動生成することが難しく, また, このようにして与えられた最適化の仕様の正当性を証明するのは困難であった．

そのようなプログラム解析の仕様は時相論理を用いることで簡潔に表現でき, その解析の自動化が容易となる．また, その枠組で記述された最適化の仕様記述が正当なものであるかの検証も, 比較的容易に行なえる [5].

Lacey らは, 時相論理 CTL に対して過去の時制を扱う演算子を加え, さらに自由変数を導入して拡張した時相論理 CTL-FV を提唱し, 伝統的なプログラム最適化の仕様の多くは, CTL-FV による条件の記述と, その結果を用いた命令文の書き換えで記述できることを示した [4]. CTL-FV は自由変数を扱うことができるので, 最適化の仕様に十分な記述力を持つ．しかし, それにより現存する一般的なモデル検査のツールで扱える範囲を越えており, CTL-FV に対するモデル検査の自動化が行ないにくい．

一方, 時相論理を用いたプログラム解析を全自動で行なう実装としては, 山岡らの解析器生成ツール [8] がある．解析対象言語を Jimple とし, Soot の Jimple パーザを用いて実装されている．Soot [7] は Java コードの最適化フレームワークであり, Java コードの解析や変換などの目的に沿ったいくつかの中間表現を生成し, またその上での基本的な操作を行なう環境を提供する．Jimple は, Soot が生成する中間表現のうちの一つで, 型付けされた 3-address 表現であり, 最適化処理を行なう際に扱うのが容易な形式をしている．[8] の実装ではモデルを構築し SMV モデルチェッカーを用いてモデル検査を行なうという一連のプログラム解析処理が, 全自動で行なわれる．しかし, この実装では, 過去時制などの拡張のない CTL に自由変数拡張を施した時相論理を性質の記述に用いている．また, 自由変数は Java の局所変数のみ取ることができ, 述語は $use()$, $def()$ のみが使えるといった, 限定されたものであった．

また, Drape らは .NET 中間表現を対象とし, 条件の記述にパス上での正規表現を用いる書き換え体系を提唱した [3]. しかし, 正規表現での記述は表記の自然さや簡潔さの面で, 時相論理での条件の記述に比べて劣る．

本論文では, CTL-FV のようなプログラム解析仕様を表すための時相論理として, NCTL+FV を提案する．NCTL+FV は時相論理 NCTL [6] に自由変数の概念を導入し拡張したものである．NCTL は CTL に対して過去の時制を扱えるように拡張されており, 性質の記述をより自然に行なうことができる．NCTL+FV は CTL-FV と比較すると, 過去時制の扱いに若干の制約があるもののプログラム解析に必要な表現能力はほとんど損なわれていない．さらに, NCTL+FV で与えられた宣言的な最適化の仕様から, 自動的に最適化の処理を行なう環境を得られる．これにより, コン

[†]東京大学大学院情報理工学系研究科数理情報学専攻, Depertment of Mathematical Informatics, The University of Tokyo
[‡]科学技術振興機構, Japan Science and Technology Agency

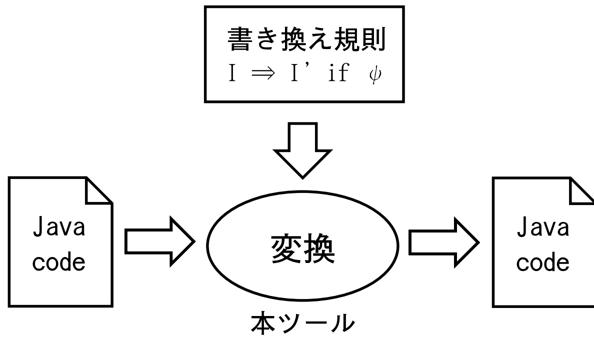


図 1: 処理概要

パイラの設計者が新たな最適化の手法を考案した時など, それを簡単にテストする環境を得られる.

また, このシステムの有効性を示すために, この枠組の上で実際に全自動でコード最適化を行なうツールを実装し, 評価実験を行なった. 最適化の規則を簡潔な仕様記述で表すことができ, 高速に実行することができた. 本実装は CTL-FV のサブセットの実装である [8] と比較すると, 過去時制の時相演算子を扱うことができ, 自由変数としてメソッド中の局所変数の他に定数, 式, 命令文, 演算子などを取ることができ, 対応してそれらの自由変数上の述語が追加されているという利点を持つ.

本論文では, 我々が実装したツールの仕様とその実装について以下の順に述べていく. 第 2 節では本ツールの概略とその仕様を述べ, 第 3 節ではその実装の詳細を述べる. 第 4 節で, 本システムの評価実験を行ない, 第 5 節で結論を述べる.

2 宣言的記述

本ツールは最適化の仕様記述をユーザから受けとり, 与えられた仕様に従って Java コードの書き換えを行ない, 結果の Java コードを出力する (図 1).

仕様記述は Jimple の命令文の書き換え規則 ($I \Rightarrow I'$) とその書き換えを行なう時に満たされるべき条件 ψ の組:

$$I \implies I' \text{ if } \psi$$

である. ここで, I, I' は, 自由変数 (Jimple の命令文の部分式にマッチするメタ変数) を含む Jimple の命令文で, ψ は時相論理で記述する.

2.1 CTL-FV による仕様記述

[4] ではこの条件 ψ の記述に時相論理 CTL-FV を用いていた. CTL-FV は時相論理 CTL に, 過去時制を扱う演算子 ($A^{\triangle}, E^{\triangle}$) と, 自由変数を導入して拡張したものである.

例えば CTL-FV では定数伝播の式の書き換え規則は以下のように表される.

$$\begin{aligned} & x := y \implies x := c \\ \text{if } & \mathrm{A}^{\triangle}(\neg def(y) \text{ W } stmt(y := c) \wedge conlit(c)) \end{aligned}$$

この意味は,「現在注目している命令文が, ある変数 y の値を別の変数 x に代入している文 ($x := y$) であり, そこから遡って以前に y にその値が定数式 ($conlit(c)$) であるような c が代入されており ($stmt(y := c)$), かつ, そこから現在の地点までで y の値が変更されない ($\neg def(y)$) ならば, 代入文を $x := c$ で書き換える」というものである.

この例のように, 書き換え規則の記述において, 書き換えの対象となる命令文の場所を視点として, 過去の性質を表現したい場合がある. このような時には時相論理中で過去時制を扱えると, 性質をより自然に表現できる.

2.2 NCTL+FV による仕様記述

本ツールでは, 時相論理 NCTL[6] をベースに自由変数の拡張を行なった時相論理を用いる. この時相論理を本論文では以下, NCTL+FV と呼ぶことにする.

NCTL+FV の定義に先立ち, 時相論理 PCTL を定義し, その下で NCTL を定義する.

2.2.1 PCTL とは

一般に良く知られている LTL, CTL, CTL*[2] などの時相論理に用いられる時間の表現は, 未来の時制のみを扱うが, 過去の時制に関する演算子を加えると仕様をより単純で自然に表現することが出来る.

PCTL [6] は CTL にそのような過去時制の拡張を行ない定義したものである[1]. 過去の構造を, 始状態から現在までの履歴変数として扱い, その上での時相演算子 (X^{-1}, S, N) が追加されている. 正式な定義は以下の 2.2.2 節で述べるが, 直観的には, X^{-1} は CTL*や

[1] PCTL を Propositional CTL の意味, あるいは Probabilistic CTL の意味で用いる文献もあるが, ここでの定義はそれらと異なる.

LTL での時相演算子 X の逆で, 現在の状態の一つ前の状態で成立する条件を表し, S は時相演算子 U の逆で, ある過去の時点までの性質の記述を表し, N は今までの履歴を忘れて, 現時点の状態から新たに開始される履歴の上で成り立つ条件を記述するのに用いる.

定義 2.1 (PCTL の構文規則) *PCTL* の構文規則は以下の通りである. 規則中の a は後述するモデルでの原子命題の要素を表す.

$$
\begin{array}{rcl}
PCTL \ni \psi & ::= & a \mid \neg\psi \mid \psi \wedge \psi \\
& \mid & \mathrm{EX}\psi \mid \mathrm{E}\psi\mathrm{U}\psi \mid \mathrm{A}\psi\mathrm{U}\psi \\
& \mid & \mathrm{X}^{-1}\psi \\
& \mid & \psi\mathrm{S}\psi \\
& \mid & \mathrm{N}\psi
\end{array}
$$

■

定義 2.2 (PCTL の糖衣構文) 構文定義中に現れないがよく用いられる結合子を省略形として以下のように定義する.

$$
\begin{array}{rcl}
\psi_1 \vee \psi_2 & \equiv & \neg(\neg\psi_1 \wedge \neg\psi_2) \\
\psi_1 \rightarrow \psi_2 & \equiv & \neg(\psi_1 \wedge \neg\psi_2) \\
\mathrm{EF}\psi & \equiv & \mathrm{E}\top\mathrm{U}\psi \\
\mathrm{AF}\psi & \equiv & \mathrm{A}\top\mathrm{U}\psi \\
\mathrm{EG}\psi & \equiv & \neg\mathrm{AF}\neg\psi \\
\mathrm{AG}\psi & \equiv & \neg\mathrm{EF}\neg\psi \\
\mathrm{AX}\psi & \equiv & \neg\mathrm{EX}\neg\psi \\
\mathrm{F}^{-1}\psi & \equiv & \top\mathrm{S}\psi
\end{array}
$$

■

2.2.2 PCTL の意味論

PCTL の意味論を述べる前にいくつかの記法を説明する.

検査する対象のモデルは, 原子命題の集合 *Prop* 上で定義される Kripke 構造 $M = \langle S, S_0, R, L\rangle$ であり,

- S はモデルの空でない状態の集合
- $S_0 \subset S$ はモデルの始状態の集合
- $R \subset S \times S$ は S 上の遷移関係
- $L : S \rightarrow 2^{Prop}$ は S 上で成り立つ原子命題の集合

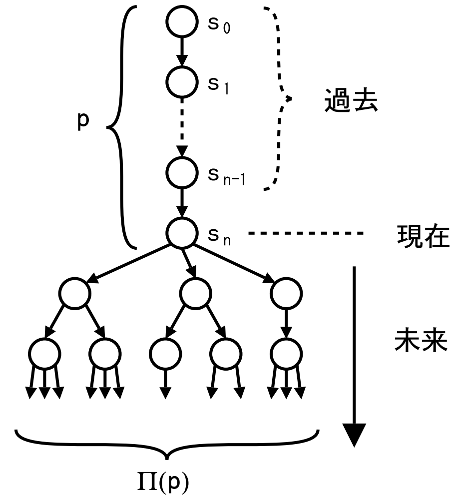


図 2: 履歴変数

で構成されている.

Kripke 構造上の **計算** (computation) とは状態の無限列 $s_0 s_1 \ldots (\forall i; s_i R s_{i+1})$ であり, π などで表す. **履歴** (history) は空でない状態の有限列 $s_0 s_1 \ldots s_n (\forall i < n; s_i R s_{i+1})$ であり, p などで表す. 計算 π, 履歴 p に対し, $\Pi(p)$ は状態列の先頭部分が p から始まる計算全ての集合を表し, $\pi_{|i}, p_{|i}$ はそれぞれ π, p のうちの先頭 $i+1$ 個の状態からなる有限列を表す. 直観的には履歴 $p = s_0 s_1 \ldots s_n$ は, 処理中の計算の, 現在の状態が s_n であり, 現在の状態に至るまでに遷移してきた状態の列が $p_{|n-1}$ であったことを示す (図 2).

定義 2.3 (PCTL の意味論) モデル M 上の履歴 $p = s_0 s_1 \ldots s_n$ のもとで 状態式 ψ が真であるということを $M, p \models \psi$ と書く. M が自明の時には省略して単に $p \models \psi$ と書く.

$$
\begin{array}{lll}
p \models a & \text{iff} & a \in L(s_n) \\
p \models \neg\psi & \text{iff} & p \models \psi \text{ でない} \\
p \models \psi_1 \wedge \psi_2 & \text{iff} & p \models \psi_1 \text{ かつ } p \models \psi_2 \\
p \models \mathrm{EX}\psi & \text{iff} & \exists\pi \in \Pi(p);\ \pi_{|n+1} \models \psi \\
p \models \mathrm{E}\psi_1\mathrm{U}\psi_2 & \text{iff} & \exists\pi \in \Pi(p); \exists m \geq n; \\
& & \pi_{|m} \models \psi_2 \text{ かつ } n \leq \forall i < m; \pi_{|i} \models \psi_1 \\
p \models \mathrm{A}\psi_1\mathrm{U}\psi_2 & \text{iff} & \forall\pi \in \Pi(p); \exists m \geq n; \\
& & \pi_{|m} \models \psi_2 \text{ かつ } n \leq \forall i < m; \pi_{|i} \models \psi_1 \\
p \models \mathrm{X}^{-1}\psi & \text{iff} & n > 0 \text{ かつ } p_{|n-1} \models \psi \\
p \models \psi_1\mathrm{S}\psi_2 & \text{iff} & \exists m \leq n; \\
& & p_{|m} \models \psi_2 \text{ かつ } m < \forall i \leq n; p_{|i} \models \psi_1 \\
p \models \mathrm{N}\psi & \text{iff} & s_n \models \psi
\end{array}
$$

モデル M で ψ が成立することを $M \models \psi$ と表し, その意味は M の始状態の全てで ψ が成立することである.

$$M \models \psi \ \text{ iff } \ \forall s \in S_0;\ M, s \models \psi$$

■

2.2.3 NCTL の構文

過去時制の時相演算子を加えたことによって, PCTL の表現力は CTL の表現力より真に大きくなっている. NCTL は PCTL の構文に制限を加えて CTL と同じ表現力を持つようにした時相論理である. NCTL の任意の式は始状態において等価な意味を持つ CTL の式に変換可能である. NCTL は CTL と同じ表現力しか持たないが, 過去時制の時相演算子を用いることで過去に関する状態の記述を CTL で記述するより簡潔に表現することが出来る.

定義 2.4 (NCTL の構文規則) *NCTL* の構文規則は以下の通りである. 規則中の a はモデルでの原子命題の要素を表す.

$$
\begin{aligned}
NCTL \ni \psi \ &::= \ \psi_{lim} \mid \neg\psi \mid \psi \wedge \psi \\
&\mid \ \mathrm{EX}\psi \mid \mathrm{E}\psi_{lim}\mathrm{U}\psi \\
&\mid \ \psi_{lim}\mathrm{S}\psi_{lim} \\
&\mid \ \mathrm{X}^{-1}\psi \\
\psi_{lim} \ &::= \ a \mid \neg\psi_{lim} \mid \psi_{lim} \wedge \psi_{lim} \\
&\mid \ \mathrm{EX}\psi_{lim} \mid \mathrm{E}\psi_{lim}\mathrm{U}\psi_{lim} \\
&\mid \ \mathrm{A}\psi_{lim}\mathrm{U}\psi_{lim} \\
&\mid \ \mathrm{F}^{-1}\psi_{lim} \\
&\mid \ \mathrm{N}\psi
\end{aligned}
$$

■

PCTL に対して A_U_ の U の両側, S の両側, E_U_ の U の左側には N をはさまない限り X^{-1}, S が現れないように制限が加えられている.

2.2.4 NCTL+FV における自由変数

NCTL+FV は NCTL に自由変数拡張を施したものである. この 自由変数は Jimple の命令文の部分式にマッチするメタ変数であり,

- ノード番号 (Jimple の命令文の番号)
- 局所変数
- 定数
- 式
- 単項演算子, 二項演算子
- 命令文

のどれかの属性が付けられている (図 3).

$$
\begin{aligned}
x, y, \ldots \ &\in \ \langle \text{局所変数} \rangle \\
c \ &\in \ \langle \text{定数} \rangle \\
e \ &\in \ \langle \text{式} \rangle \\
n, m, \ldots \ &\in \ \langle \text{ノード番号} \rangle \\
\otimes \ &\in \ \langle \text{二項演算子} \rangle \\
\ominus \ &\in \ \langle \text{単項演算子} \rangle \\
I \ &\in \ \langle \text{命令文} \rangle
\end{aligned}
$$

$$
\begin{aligned}
I \ &::= \ \mathrm{skip} \mid x := e \mid \mathrm{return} \mid \mathrm{return}\ e \mid \ldots \\
e \ &::= \ x \mid c \mid \ominus e \mid e \otimes e \mid \ldots \\
n \ &::= \ 0 \mid 1 \mid \ldots
\end{aligned}
$$

図 3: 自由変数の属性

また, NCTL+FV の原子命題は自由変数上の述語に拡張されており, 条件文 ψ 中で, Jimple 命令文の部分式を引用することができる.

述語としては以下のものが用意されている.

true		常に真
use(x)	iff	s_nで局所変数 x が参照されている
def(x)	iff	s_nで局所変数 x に代入されている
stmt(I)	iff	s_nが I の形の命令文である
node(n)	iff	s_nのノード番号が n である
trans(e)	iff	s_nで式 e の中に含まれる変数が変更されない

2.3 最適化処理の記述

本節で述べた NCTL+FV の仕様を用いた最適化の記述の例を挙げる.

定数伝播 以下のような二命令を含む命令列を考える.

(i) a := 10
 ⋮
(ii) b := a

この命令列において,

- (ii) 式の前に必ず (i) が必ず実行されること (コントロールフロー解析)
- (i) 式から (ii) の間に 変数 a の値が変更されないこと (データフロー解析)

のそれぞれがプログラム解析で確認できるならば, (ii) 式中の a を

(ii') b := 10

のように置き換えて構わない．このような変換を定数伝播の適用という．これは以下のような仕様で記述できる．

$$x := y \implies x := c \ \text{ if } \ \neg \mathit{def}(y)\ \mathrm{S}\ \mathit{stmt}(y := c) \quad (1)$$

この意味は，「現在注目している命令文が，ある変数 y から別の変数 x への代入文 ($x := y$) であり，ある定数 c に対してそこから遡って以前に y に c が代入されており ($\mathit{stmt}(y := c)$)，かつ，そこから現在の地点までの間に y の値が変更されない ($\neg \mathit{def}(y)$) ならば，代入文を $x := c$ で書き換える」というものであり，2.1 節に挙げた仕様を NCTL+FV で記述した例である．

このように，CTL-FV で記述された条件式の多くは NCTL+FV でも記述することができる．NCTL+FV の構文上の制約より，例えば S が入れ子になる式は記述できないが，そのような式が有用な最適化の条件式に必須であるかどうかは不明である．

定数の畳み込み　式が定数同士の基本演算である時は，その値の計算をあらかじめ行なうことができ，その値で置き換える．

$$x := c_1 \otimes c_2 \implies x := \mathit{eval}(c_1 \otimes c_2)$$

$$x := \ominus\ c \implies x := \mathit{eval}(\ominus\ c)$$

不要代入命令除去　代入文の後で，代入された値が一度も参照されないならば，その代入文は不要であるので除去することができる．

$$\begin{array}{l} x := e \implies \text{skip} \\ \text{if } \ \mathrm{AX}\big(\mathrm{AG}\neg \mathit{use}(x) \ \vee \ \mathrm{A}\neg \mathit{use}(x) \mathrm{U}\, \mathit{def}(x)\big) \end{array} \quad (2)$$

ループ不変式の巻き上げ　ループ中で計算されている式の値が，ループ中では変化しない値ならば，ループを回るたびに計算しても無駄であるので，その式の計算をループ前に移動する．

$$\begin{array}{rl} m: & I \implies I\ ;\ x := e \\ n: & x := e \implies \text{skip} \\ \text{if } \ n: & \mathrm{EXE}\neg \mathit{node}(m)\ \mathrm{U}\ \mathit{node}(n) \\ & \wedge\Big(\mathit{trans}(e) \wedge \big(\neg \mathit{def}(x) \vee \mathit{node}(n)\big)\Big)\ \mathrm{S}\ \mathit{node}(m) \end{array} \quad (3)$$

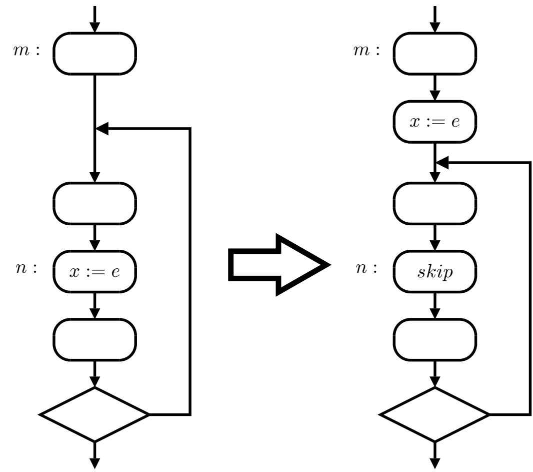


図 4: ループ不変式の巻き上げ

この仕様の条件式は論理式の吸収則 ($\mathrm{F}^{-1}\psi_2 \wedge \psi_1 \mathrm{S} \psi_2 \equiv \psi_1 \mathrm{S} \psi_2$) より

$$\begin{array}{l} \mathrm{F}^{-1}\mathit{node}(m) \wedge \mathrm{EXE}\neg \mathit{node}(m)\ \mathrm{U}\ \mathit{node}(n) \\ \wedge\Big(\mathit{trans}(e) \wedge \big(\neg \mathit{def}(x) \vee \mathit{node}(n)\big)\Big)\ \mathrm{S}\ \mathit{node}(m) \end{array}$$

と等価であるが，この一行目は「n に到達したときには必ず m を通っていて，かつ，m を経由せず再度 n に到達する経路が存在する」ということを表す．すなわち図 4 のように n を含むループが存在し，その外側に m が存在することを表す．二行目は「n を含むループと，m から n への経路上のどちらにも式 e に含まれる各変数と変数 x に変更が加えられていない」ということを表す．

3　本ツールの実装

本ツールは図 1 のように，入力として最適化の仕様 ($I \Rightarrow I'$ if ψ) と，Java のコードを受けとる．与えられた最適化の仕様に基づいた変換を施し，変換後の Java コードを出力する．

1. まず入力された仕様を解析する．
2. 仕様中の NCTL+FV 式 ψ から過去時制を取り除き，自由変数拡張をした CTL(以下，CTL+FV と呼ぶ) の式 ψ' に変換する．
3. そして，Soot に処理を渡し，Java コードから Jimple 中間表現を得る．

その後 Soot の Jimple 変換パス (jtp) 中に，対象コードのメソッドそれぞれに対して，本ツールの中心

部分である解析変換処理部分が呼び出される．解析変換処理部分は与えられたメソッドに対して以下の順に処理を行なう．

1. メソッドのコントロールフローグラフからモデル検査で用いるためのモデルの一部を構成する．
2. メソッドの各命令文に対して I とマッチするかどうかを検査する．
3. 2 を満たしたものについて，自由変数の割り当てを行ないモデル M 全体を構成し，モデル検査 $M \models \psi'$ を行なう．
4. 3 を満たしたものについて，Soot の Jimple の命令文操作処理を用いて Jimple 上での Java コードの書き換えを行なう．

以下，これらの処理を順に説明していく．

3.1 仕様の解釈

与えられた仕様

$$I \implies I' \text{ if } \psi$$

に対して，暗黙的に行への参照を付加する．

$$n : I \implies I' \text{ if } \psi$$

これらは内部的に

if $s_n \models stmt(I)$ **and** $M \models AG(node(n) \rightarrow \psi)$
then rewrite I to I'

という処理に解釈される．

3.2 NCTL+FV から CTL+FV への変換

与えられた ψ が過去時制を含む時は，それを過去時制を含まない等価な CTL+FV の式 ψ' に書き換える処理を行なう．NCTL で与えられた仕様を等価な CTL に変換する規則は [6] で述べられており (巻末 A 節参照)，本ツールはこの変換規則を実装して用いている．これにより NCTL+FV の式を CTL+FV に変換する．一般にこの変換を行なうことによって，NCTL+FV の論理式 ψ の長さに対して，変換後の CTL+FV の論理式 ψ' の長さが組合せ的に爆発する可能性がある．本ツールに実装されている変換では，変換時に構文木の部分木を共有することにより共通な部分式の計算をまとめ，爆発をある程度抑える．

```
    int sum(int)
    {
        Sum r0;
        int i0, i1, $i2;

0:      r0 := @this: Sum;
1:      i0 := @parameter0: int;
2:      i1 = 0;
3:      if i0 >= 0 goto label0;

4:      return i1;

     label0:
5:      if i0 < 0 goto label1;

6:      $i2 = i0;
7:      i0 = i0 + -1;
8:      i1 = i1 + $i2;
9:      goto label0;

     label1:
10:     return i1;
    }
```

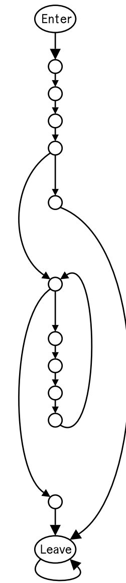


図 7: コントロールフローグラフからモデルを構成

図 5 に式 (1) の定数伝播の性質記述 ($\psi = \neg def(y)$ S $stmt(y := c)$) の変換例を挙げる．

この変換を経て，変換後の式が長くなっているが，同じ部分式が何度も出現している．これらをまとめ上げ，最終的に図 6 の内部表現が得られる．この図の例では $10, 11, 13, 14, 17$ の節が共有されている．

3.3 モデルの作成

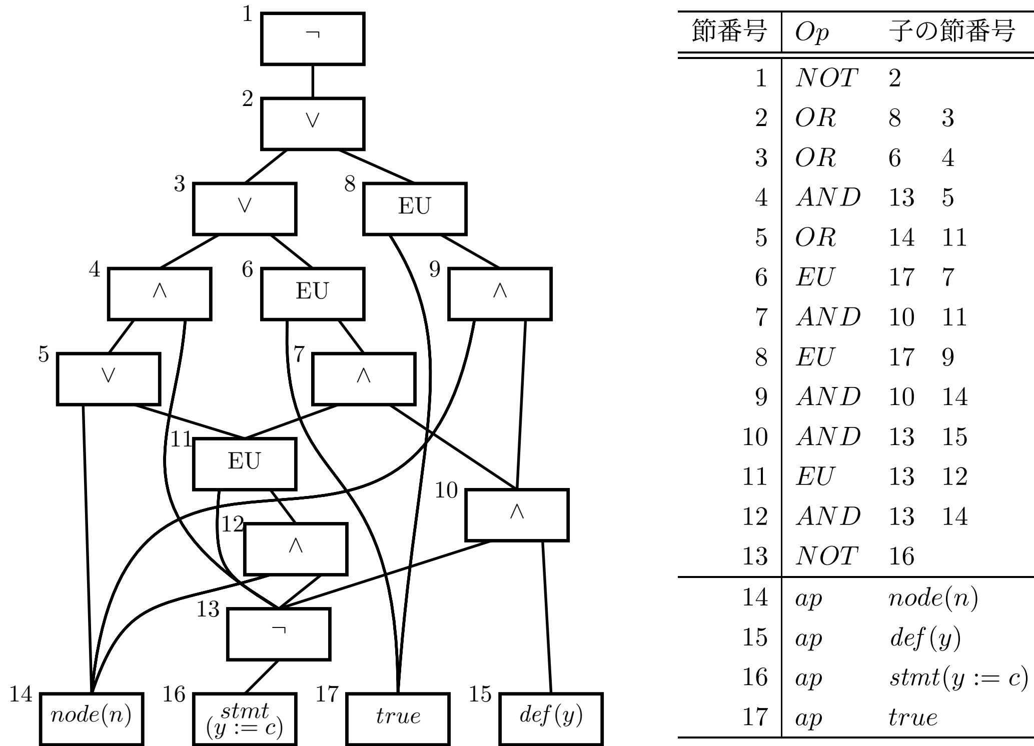
メソッドのコントロールフローグラフから図 7 のように，モデル $M = \langle S, S_0, R, L\rangle$ のうち S, S_0, R を構成する．

- S の各要素は，メソッド中の Jimple の各命令文と，そこに “Enter”，“Leave” の仮想的な二つのノードを加えたものとする．
- S_0 は “Enter” ノードとする．
- R はメソッド中の Jimple の各命令文の間の遷移関係とする．仮想的に加えた二つのノードについては，“Enter” からの遷移はメソッドの入口へとし，“Leave” への遷移はメソッドの脱出口からと，“Leave” 自身からである．

$$
\begin{aligned}
& \mathrm{AG}\Big(\mathit{node}(n) \rightarrow \big(\neg \mathit{def}(y)\ \mathrm{S}\ \mathit{stmt}(y := c)\big)\Big) \\
\equiv\ & \neg \mathrm{EF}\Big(\mathit{node}(n) \wedge \neg\big(\neg \mathit{def}(y)\ \mathrm{S}\ \mathit{stmt}(y := c)\big)\Big) \\
\equiv\ & \neg\Bigg(\mathrm{EF}\big(\mathit{node}(n) \wedge \mathit{def}(y) \wedge \neg \mathit{stmt}(y := c)\big) \\
& \quad \vee\ \mathrm{EF}\Big(\mathit{def}(y) \wedge \neg \mathit{stmt}(y := c) \wedge \mathrm{E}\big(\neg \mathit{stmt}(y := c)\big)\mathrm{U}\big(\mathit{node}(n) \wedge \neg \mathit{stmt}(y := c)\big)\Big) \\
& \quad \vee\ \neg\big(\neg \mathit{def}(y)\ \mathrm{S}\ \mathit{stmt}(y := c)\big) \wedge \Big(\mathit{node}(n) \vee \mathrm{E}\big(\neg \mathit{stmt}(y := c)\big)\mathrm{U}\big(\mathit{node}(n) \wedge \neg \mathit{stmt}(y := c)\big)\Big)\Bigg) \\
\equiv_i\ & \neg\Bigg(\mathrm{EF}\big(\mathit{node}(n) \wedge \mathit{def}(y) \wedge \neg \mathit{stmt}(y := c)\big) \\
& \quad \vee\ \mathrm{EF}\Big(\mathit{def}(y) \wedge \neg \mathit{stmt}(y := c) \wedge \mathrm{E}\big(\neg \mathit{stmt}(y := c)\big)\mathrm{U}\big(\mathit{node}(n) \wedge \neg \mathit{stmt}(y := c)\big)\Big) \\
& \quad \vee\ \neg \mathit{stmt}(y := c) \wedge \Big(\mathit{node}(n) \vee \mathrm{E}\big(\neg \mathit{stmt}(y := c)\big)\mathrm{U}\big(\mathit{node}(n) \wedge \neg \mathit{stmt}(y := c)\big)\Big)\Bigg)
\end{aligned}
$$

式中で $\equiv_i$ は, 始状態において等号の前後の式が同値であることを表す.

図 5: 過去時制の除去



節番号	*Op*	子の節番号	
1	*NOT*	2	
2	*OR*	8	3
3	*OR*	6	4
4	*AND*	13	5
5	*OR*	14	11
6	*EU*	17	7
7	*AND*	10	11
8	*EU*	17	9
9	*AND*	10	14
10	*AND*	13	15
11	*EU*	13	12
12	*AND*	13	14
13	*NOT*	16	
14	*ap*	$\mathit{node}(n)$	
15	*ap*	$\mathit{def}(y)$	
16	*ap*	$\mathit{stmt}(y := c)$	
17	*ap*	*true*	

図 6: 定数伝播の論理式の CTL 構文木表現

3.4 自由変数の束縛

メソッドに含まれる Jimple の各命令文に対して, I とマッチするか否かを検査する. マッチ σ が存在する場合, I 中に現れず ψ 中にのみ現れる自由変数全てを対象メソッド内で現れた要素で束縛し, それぞれをモデル検査する.

各述語の自由変数に, 属性に合わせて具体的な Jimple 中の要素 (局所変数, 定数, ...) を割り当てることにより, 各述語が具体化される (図 8). この時に, 例えば定数の属性を持つ自由変数には, メソッド中に現れた全ての定数の割り当てを試す. S の各状態での述語の値を計算することにより Kripke 構造の原子命題集合 L を構成でき, 3.3 節の結果とあわせて $M = \langle S, S_0, R, L\rangle$ が構成される.

疑似コードで表すと以下のようになる.

foreach $(n : i) \in \mathit{method}$ **do**
  foreach $\sigma \in \mathit{matches}(i, I)$ **do**
    if $\exists\sigma' \supset \sigma; M_{\sigma'} \models \psi'$ **then**
      $\mathit{delete}(n)$
      $\mathit{insert}(n, \sigma' I')$
    endif
  done
done

例として式 (1) の仕様と図 7 のメソッドに関しての処理を考える. 図中, 6: の行が $n : x := y$ の形式にマッチし, $\sigma = \{n \mapsto 6,\ x \mapsto \$i2,\ y \mapsto i0\}$ を得る. ψ 中に現れる自由変数 c は, I 中に含まれないので σ に含まれていない. 次に σ を含み, c を要素に持つマッチ σ' を全て考える. メソッド中に現れる定数は $0, -1$ のみなので, σ' の取り得るマッチは

$$\sigma' \in \left\{ \begin{array}{l} \{n \mapsto 6,\ x \mapsto \$i2,\ y \mapsto i0,\ c \mapsto 0\}, \\ \{n \mapsto 6,\ x \mapsto \$i2,\ y \mapsto i0,\ c \mapsto -1\} \end{array} \right\}$$

の要素全てである. これらの各々について σ' の下でのモデル $M_{\sigma'}$ を構成し, モデル検査を行なう.

3.5 モデル検査

ここまでで $M_{\sigma'} = \langle S, S_0, R, L\rangle$ を構成することができたので, モデル検査を行ない $M_{\sigma'} \models \psi'$ が成立するかどうかを求める.

本実装では [1] に基づき古典的な CTL モデル検査のアルゴリズムを用いており, 計算量は $O(|\psi'| \times (|S| + |R|))$ である. ここで, $|\psi'|$ は論理式 ψ' の長さであり[2], $|S|$ は S に含まれる状態の個数, すなわち解析対象メソッド中の命令文の数+2, $|R|$ は R に含まれる遷移の数の合計である.

14	ap	$\mathit{node}(n)$
15	ap	$\mathit{def}(y)$
16	ap	$\mathit{stmt}(y := c)$
17	ap	true

$$\Downarrow \quad \sigma' = \{n \mapsto 6,\ x \mapsto \$i2,\ y \mapsto i0,\ c \mapsto -1\}$$

14	ap	$\mathit{node}(6)$
15	ap	$\mathit{def}(i0)$
16	ap	$\mathit{stmt}(i0 := -1)$
17	ap	true

図 8: 原子命題の自由変数を束縛

3.6 命令文書き換え

もし, モデル検査の結果, $M_{\sigma'} \models \psi'$ が成立したならば, 規則 $I \Rightarrow I'$ に基づき命令文の書き換えが行なわれる. 現在対象としている行を削除し, 代わりに I' 中の自由変数を σ' で置き換えたものを現在の位置に挿入する.

3.7 計算量

このツールの処理時間の大半はモデル検査に消費される. 3.5 節で, モデル検査の 1 回の計算量が $O(|\psi'| \times (|S| + |R|))$ であることを述べた. モデル検査が実行される回数は, 3.4 節の自由変数の束縛時の σ' の取り得る組合せの個数を $|\sigma'|$, 書き換え規則 $I \Rightarrow I'$ の個数を $|I|$ とするとき[3], $O(|S|^{|I|} \times |\sigma'|)$ 回のモデル検査が行なわれる. 以上から, 全モデル検査の計算量は $O(|\psi'| \times (|S| + |R|) \times |S|^{|I|} \times |\sigma'|)$ となる[4]. [8] の実装では, 条件式中に現れた全ての自由変数に対して組合せを試しているのに対し, この実装では I 中に含まれず, ψ 中にのみ現れる自由変数に対しての組合せを

[2] より正確には ψ' を構文木で表した時の総節点数である.

[3] 例えば, 2.3 節の不要命令除去 (式 (1)) と, 定数伝播 (式 (2)) の仕様では $|I| = 1$, ループ不変式の巻き上げ (式 (3) では $|I| = 2$ である

[4] switch 文以外では二ヶ所より多くに分岐しないため, 通常 $|R| = \Theta(|S|)$ が成立し, $O(|\psi'| \times |S|^{|I|+1} \times |\sigma'|)$ となる.

表 1: 実験結果 1 (実際のコードへの適用)

対象	org-apache	BigInteger
クラスファイル数	41	1
ファイルサイズ [bytes]	104,147	30,467
メソッド数	426	105
Soot の消費時間 [秒]	14.0	12.9
不用命令除去		
除去した命令数	20	13
要した時間 [秒]*	1.6(15.6)	1.2(14.1)
定数伝播		
変換した命令数	0	3
要した時間 [秒]*	1.7(15.7)	1.9(14.8)

* 括弧の内側の数値は処理全体の時間, 括弧の外側の数値は Soot 自体の処理時間を差し引いた時間である.

試すだけなので, モデル検査を行なう回数を大幅に削減できる.

4 実験

実際に以下の Java コードに対して, 2.3 節の不要命令除去 (式 (1)) と, 定数伝播 (式 (2)) の最適化を行ない, どの程度の時間がかかるか計測を行なった.

- org-apache
  空のクラス Nothing と, その実行時に要求されるランタイムライブラリ群のうち, `org.apache` 下のファイル全体. これは標準の実行環境に用いられるライブラリに, 最適化の余地があるかどうかを調べることにつながる. Soot の `-app -i org.apache` オプションを用いた.
- BigInteger
  java.math.BigInteger クラス. クラスファイルサイズが比較的大きい.

実験に用いた環境は以下の通りである.

- Soot 2.0.1
- 実行環境: Blackdown Java-Linux Version 1.4.1-beta
- CPU: Pentium M 1.3GHz
- メモリ: 512Mbytes

結果を表 1 に示す. 数十から百 kbytes 程度のクラスファイルの解析でも, 最適化処理が数秒程度で終了していることがわかる.

表 2: 実験結果 2 (論理式の性質と時間との関係)

種類	$\|\psi'\|$	自由変数の個数*	要した時間 [秒]**	
Soot				7.4
1	10	0	0.4	(7.8)
2	10	定数 ×1	2.3	(9.7)
3	10	定数 ×2	20.2	(27.6)
4	10	定数 ×3	197.7	(205.1)
5	10	局所変数 ×1	39.9	(47.3)
6	1000	0	23.1	(30.5)

* ψ 中にのみ現れ, I 中に現れない自由変数の属性の個数.
** 括弧の内側の数値は処理全体の時間, 括弧の外側の数値は Soot 自体の処理時間を差し引いた時間である.

どちらの対象コードも不用命令を検出と除去を行ない, BigInteger については定数伝播の検出と最適化を行なった. 広く流布しているライブラリのコードの最適化し尽くされていない部分を検出し, その部分に対して最適化を施すことができた.

定数伝播の適用が不用命令除去に比べて処理時間がかかっているのは, 不用命令除去のほうは ψ のみに含まれる自由変数が存在しないのに対し, 定数伝播のほうは定数の属性を持つ自由変数を一つ持つためである.

また, 対象とするクラスを固定し, 論理式の長さや自由変数の個数を変化させて時間を計測し, これらの個数の処理時間への影響を調べた.

- TimeTest
  このクラスは 2 つのメソッドを持ち, 一つは小さなインスタンス初期化メソッドであり, もう一つは, 定数を 10 個, 局所変数 200 個持つ, 命令文数 202 個のメソッドである.

表 2 に論理式の長さや自由変数の個数に対する処理時間を示す. 束縛を試す自由変数が増えるにしたがって, 3.7 節の評価に近い処理時間の増加が見られるのが分かる. ただし先ほどの定数伝播のように, 実際の最適化に仕様においては, 束縛が必要な変数の個数が少なく, 処理時間が問題にならない程度で済むことが多い.

5 おわりに

本論文では, 与えられた仕様に基づき自動的にプログラム変換を行なうツールの設計と実装を述べた. この仕様記述のために, 時相論理 NCTL+FV を提案し

た. NCTL+FV は, 自由変数を扱えるように時相論理 NCTL を拡張したものであり, 変数は Jimple 上の局所変数, 定数, 式 など属性が割り当てられる.

本ツールは Java 最適化フレームワーク Soot 上で実装されており, NCTL+FV から CTL+FV への変換器を経由して組み込みの CTL モデル検査器を用いてモデル検査を行ない, その結果を用いて Jimple 上の命令文の書き替えを行なう. 最適化処理は高速に行なうことができ, 広く利用されているコードの最適化漏れを検出し, 修正を行なうことが可能である.

通常のコンパイラの最適化においては, 最適化を施す順序により最適化の質が変わってしまうため, 適用順序の選択には工夫がなされているが, 本ツールの現在の実装では, 複数の仕様を与えても, それらを与えられた順に適用可能かどうかを調べるだけとなっている.

NCTL+FV による仕様記述を用いて, いくつかのコンパイラの最適化の記述を行なうことができることを示したが, 現在の枠組では扱えない最適化も存在する. 例えば, ループ展開など, ブロック単位での複写を必要とするものは, 現在の枠組を越えているが, ブロックにマッチする自由変数を導入することで解決できる. また, 現在は関数内解析のみしか行なえないため, 関数間にまたがる最適化を扱うことができない. Soot が持つ関数間解析の枠組を用いて, 関数間にまたがる最適化を扱う枠組に拡張することが望まれる. また, より多くの例を調査し, この枠組で扱うことができる最適化の範囲を明確にする必要がある. さらに, ユーザーが新たな述語を必要とした時に, それを容易に追加することができるような仕組みを実装し, より汎用的なシステムにしたいと考えている.

本実装では組み込みのモデル検査器を用意し, それを使ってモデル検査を行なっているが, この検査器はあまり効率が良いものではない. この点を改善するために, 組み込みのモデル検査器ではなく, SMV などの既存の優秀なモデル検査器を利用することも考えられる. しかしその際には, 本実装が NCTL+FV から CTL+FV への変換時に行なっている, 部分木の共有による式の圧縮が無効になるため, 効率の低下を招くかもしれない. また, 別の方向性としては, 現在の組み込みのモデル検査器を, BDD などを用いて高速処理を行なえるように改良することも考えられる.

現在の実装は, 自由変数の解決を行なう際に, その属性に対応する, メソッド中に出現した全要素の割り当てを試している. そのため条件式に含まれる自由変数が多くなると, 処理時間が指数関数的に増大する. 条件式を解析し, 試す割り当てに制限を加えることで, 処理時間を削減する必要がある.

また, 過去時制を扱うために, 条件式の記述に用いる時相論理を NCTL+FV とし, NCTL+FV から CTL+FV への変換を経由しモデル検査を行なっているが, この変換において, 変換処理の高速化や, 生成される CTL+FV 論理式の長さを効率化する処理はあまり行なっていない. これらの処理の効率化は今後の重要な課題である.

謝辞

本論文の執筆にあたり有益な助言を下さった査読者の方々に感謝致します.

参考文献

[1] Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM transactions on Programming Languages and Systems*, Vol. 8, No. 2, pp. 244–263, 1986.

[2] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 1999.

[3] Stephen Drape, Oege de Moor, and Ganesh Sittampalam. Transforming the .NET intermediate language using path logic programming. In *Proceedings of the 4th ACM SIGPLAN international conference on Principles and practice of declarative programming*, pp. 133–144, 2002.

[4] David Lacey and Oege de Moor. Imperative program transformation by rewriting. In R. Wilhelm, editor, *Proceedings of the 10th International Conference on Compiler Construction*, Vol. 2027 of *LNCS*, pp. 52–68. Springer Verlag, 2001.

[5] David Lacey, Neil D. Jones, Eric Van Wyk, and Carl Christian Frederiksen. Proving correctness of compiler optimizations by temporal logic. In

Proceedings of Symposium on Principles of Programming Languages, pp. 283–294, 2002.

[6] François Laroussinie and Philippe Schnoebelen. Specification in CTL+Past for verification in CTL. *Information and Computation*, Vol. 156, No. 1/2, pp. 236–263, 2000.

[7] Raja Vallée-Rai, Laurie Hendren, Vijay Sundaresan, Patrick Lam, Etienne Gagnon, and Phong Co. Soot - a java optimization framework. In *Proceedings of CASCON 1999*, pp. 125–135, 1999.

[8] 山岡裕司, 胡振江, 武市正人, 小川瑞史. モデル検査技術を利用したプログラム解析器の生成ツール. 情報処理学会論文誌, Vol. 44, No. SIG13(PRO18), pp. 25–37, 2003.

A　NCTL の分離定理

NCTL の分離定理を [6] から引用する．この変換によって, 未来時制の演算子の中にある過去時制の演算子を外側に掃き出す.

$$
\begin{aligned}
&(R1) && \mathrm{E}\phi\mathrm{U}(\alpha \wedge \mathrm{X}^{-1}x) && \equiv \big(\alpha \wedge \mathrm{X}^{-1}x\big) \vee \big(\mathrm{E}\phi\mathrm{U}(\phi \wedge x \wedge \mathrm{EX}\alpha)\big) \\
&(R2) && \mathrm{E}\phi\mathrm{U}(\alpha \wedge \neg\mathrm{X}^{-1}x) && \equiv \big(\alpha \wedge \neg\mathrm{X}^{-1}x\big) \vee \big(\mathrm{E}\phi\mathrm{U}(\phi \wedge \neg x \wedge \mathrm{EX}\alpha)\big) \\
&(R3) && \mathrm{EX}(\alpha \wedge \mathrm{X}^{-1}x) && \equiv x \wedge \mathrm{EX}\alpha \\
&(R4) && \mathrm{EX}(\alpha \wedge \neg\mathrm{X}^{-1}x) && \equiv \neg x \wedge \mathrm{EX}\alpha \\
&(R5) && \mathrm{E}\phi\mathrm{U}(\alpha \wedge x\mathrm{S}y) && \equiv \mathrm{E}\phi\mathrm{U}(\alpha \wedge y) \vee \mathrm{E}\phi\mathrm{U}\Big(\phi \wedge y \wedge \mathrm{EXE}(\phi \wedge x)\mathrm{U}(\alpha \wedge x)\Big) \\
&&&&& \quad \vee \Big(x\mathrm{S}y \wedge \big(\alpha \vee \mathrm{E}(\phi \wedge x)\mathrm{U}(\alpha \wedge x)\big)\Big) \\
&(R6) && \mathrm{E}\phi\mathrm{U}\big(\alpha \wedge \neg(x\mathrm{S}y)\big) && \equiv \mathrm{E}\phi\mathrm{U}(\alpha \wedge \neg x \wedge \neg y) \vee \mathrm{E}\phi\mathrm{U}\Big(\phi \wedge \neg x \wedge \neg y \wedge \mathrm{E}(\phi \wedge \neg y)\mathrm{U}(\alpha \wedge \neg y)\Big) \\
&&&&& \quad \vee \Big(\neg(x\mathrm{S}y) \wedge \big(\alpha \vee \mathrm{E}(\phi \wedge \neg y)\mathrm{U}(\alpha \wedge \neg y)\big)\Big) \\
&(R7) && \mathrm{EX}(\alpha \wedge x\mathrm{S}y) && \equiv \mathrm{EX}(\alpha \wedge y) \vee \big(x\mathrm{S}y \wedge \mathrm{EX}(\alpha \wedge x)\big) \\
&(R8) && \mathrm{EX}\big(\alpha \wedge \neg(x\mathrm{S}y)\big) && \equiv \mathrm{EX}(\alpha \wedge \neg x \wedge \neg y) \vee \Big(\neg(x\mathrm{S}y) \wedge \mathrm{EX}(\alpha \wedge \neg y)\Big) \\
&(R9) && \mathrm{E}\Big((\mathrm{F}^{-1}x \wedge \alpha) \vee (\neg\mathrm{F}^{-1}x \wedge \beta) \vee \gamma\Big)\mathrm{U}\Big((\mathrm{F}^{-1}x \wedge \alpha') \vee (\neg\mathrm{F}^{-1}x \wedge \beta') \vee \gamma'\Big) \\
&&&&& \equiv \mathrm{F}^{-1}x \wedge \mathrm{E}(\alpha \vee \gamma)\mathrm{U}(\alpha' \vee \gamma') \\
&&&&& \quad \vee \neg\mathrm{F}^{-1}x \wedge \mathrm{E}\Big(\neg x \wedge (\beta \vee \gamma)\Big)\mathrm{U}\Big(x \wedge \mathrm{E}(\alpha \vee \gamma)\mathrm{U}(\alpha' \vee \gamma')\Big) \\
&&&&& \quad \vee \neg\mathrm{F}^{-1}x \wedge \mathrm{E}\Big(\neg x \wedge (\beta \vee \gamma)\Big)\mathrm{U}\Big(\neg x \wedge (\beta' \vee \gamma')\Big) \\
&(R10) && \mathrm{EG}\Big((\mathrm{F}^{-1}x \wedge \alpha) \vee (\neg\mathrm{F}^{-1}x \wedge \beta) \vee \gamma\Big) \\
&&&&& \equiv \mathrm{F}^{-1}x \wedge \mathrm{EG}(\alpha \vee \gamma) \\
&&&&& \quad \vee \neg\mathrm{F}^{-1}x \wedge \mathrm{E}\Big(\neg x \wedge (\beta \vee \gamma)\Big)\mathrm{U}\Big(x \wedge \mathrm{EG}(\alpha \vee \gamma)\Big) \\
&&&&& \quad \vee \neg\mathrm{F}^{-1}x \wedge \mathrm{EG}\Big(\neg x \wedge (\beta \vee \gamma)\Big) \\
&(R11) && \mathrm{EX}(\alpha \wedge \mathrm{F}^{-1}x) && \equiv \mathrm{EX}(\alpha \wedge x) \vee (\mathrm{F}^{-1}x \wedge \mathrm{EX}\alpha) \\
&(R12) && \mathrm{EX}\big(\alpha \wedge \neg\mathrm{F}^{-1}x\big) && \equiv \neg\mathrm{F}^{-1}x \wedge \mathrm{EX}(\alpha \wedge \neg x)
\end{aligned}
$$