CS 511, Fall 2018, Handout 25

Program Schemes and First-Order Logic

Assaf Kfoury

1 November 2018

PROGRAMS and PROGRAM SCHEMES

- Let P be a program in some program language (*e.g.*, Python, Java, Haskell, C, etc.).
- P uses several primitive operators ("*prim ops*") (*e.g.*, $+$, $\times$, $\div$, div, mod, $<=$, $!=$, etc.)
- P operates over one or several domains (*e.g.*, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{B}$, etc.)

PROGRAMS and PROGRAM SCHEMES

- Let P be a program in some program language (*e.g.*, Python, Java, Haskell, C, etc.).
- P uses several primitive operators ("*prim ops*") (*e.g.*, $+$, $\times$, $\div$, div, mod, $<=$, $!=$, etc.)
- P operates over one or several domains (*e.g.*, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{B}$, etc.)

- We obtain a program scheme S from P by omitting the meaning of all the prim ops and leaving them as uninterpreted functions and uninterpreted relations.
- S is thus the part of P that directs execution according to P's code, *i.e.*, S can be viewed as P's control structure which determines P's flow of execution.
- We recover P from S by restoring the meaning of all the prim ops.

PROGRAMS and PROGRAM SCHEMES

- Let P be a program in some program language (*e.g.*, Python, Java, Haskell, C, etc.).
- P uses several primitive operators ("*prim ops*") (*e.g.*, $+$, $\times$, $\div$, div, mod, $<=$, $!=$, etc.)
- P operates over one or several domains (*e.g.*, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{B}$, etc.)

- We obtain a program scheme S from P by omitting the meaning of all the prim ops and leaving them as uninterpreted functions and uninterpreted relations.
- S is thus the part of P that directs execution according to P's code, *i.e.*, S can be viewed as P's control structure which determines P's flow of execution.
- We recover P from S by restoring the meaning of all the prim ops.

- Some of the material to follow in this handout is closely related to Handout 12 on Unwinding of Programs.

example: a PROGRAM P and corresponding PROGRAM SCHEME S

Euclidean GCD program

precondition :

$x > 0$ and $y > 0$

```
m := min(x, y)
n := max(x, y)
while m ≠ 0
    r := (n mod m)
    n := m
    m := r
return n
```

example: a PROGRAM P and corresponding PROGRAM SCHEME S

Euclidean GCD program

precondition :

$x > 0$ and $y > 0$

$m := \min(x, y)$
$n := \max(x, y)$
while $m \neq 0$
    $r := (n$ mod $m)$
    $n := m$
    $m := r$
return n

corresponding program scheme

precondition :

$\mathbf{R}\,(x, \mathbf{c}\,) \;\wedge\; \mathbf{R}\,(y, \mathbf{c}\,)$

$m := \mathbf{lo}\,(x, y)$
$n := \mathbf{hi}\,(x, y)$
while $\neg(m \doteq \mathbf{c}\,)$
    $r := \mathbf{f}\,(n, m)$
    $n := m$
    $m := r$
return n

example: a PROGRAM P and corresponding PROGRAM SCHEME S

Euclidean GCD program

precondition :

$x > 0$ and $y > 0$

1 : $m := \min(x, y)$
2 : $n := \max(x, y)$
3 : **while** $m \neq 0$
4 : $\quad r := (n \bmod m)$
5 : $\quad n := m$
6 : $\quad m := r$
7 : **return** n

corresponding program scheme

precondition :

$\mathbf{R}(x, \mathbf{c}) \wedge \mathbf{R}(y, \mathbf{c})$

1 : $m := \mathbf{lo}(x, y)$
2 : $n := \mathbf{hi}(x, y)$
3 : **while** $\neg(m \doteq \mathbf{c})$
4 : $\quad r := \mathbf{f}(n, m)$
5 : $\quad n := m$
6 : $\quad m := r$
7 : **return** n

Program execution is fully determined by the values of input variables x and y, *i.e.*, by constraints exclusively involving x and y and none of the program/internal variables $\{m, n, r\}$, *e.g.*, consider the number of times the loop body $\{4, 5, 6\}$ is executed.

example: a PROGRAM P and corresponding PROGRAM SCHEME S

Euclidean GCD program

precondition :

$x > 0$ and $y > 0$

1 : $m := \min(x,y)$

2 : $n := \max(x,y)$

3 : **while** $m \neq 0$

4 : $\quad r := (n \bmod m)$

5 : $\quad n := m$

6 : $\quad m := r$

7 : **return** n

corresponding program scheme

precondition :

$\mathbf{R}\,(x,\mathbf{c}\,) \wedge \mathbf{R}\,(y,\mathbf{c}\,)$

1 : $m := \mathbf{lo}\,(x,y)$

2 : $n := \mathbf{hi}\,(x,y)$

3 : **while** $\neg(m \doteq \mathbf{c}\,)$

4 : $\quad r := \mathbf{f}\,(n,m)$

5 : $\quad n := m$

6 : $\quad m := r$

7 : **return** n

Program execution is fully determined by the values of input variables x and y, *i.e.*, by constraints exclusively involving x and y and none of the program/internal variables $\{m,n,r\}$, *e.g.*, consider the number of times the loop body $\{4,5,6\}$ is executed.

For example, $\{4,5,6\}$ is executed twice iff:

$\min(x,y) \neq 0$ &

$\max(x,y) \bmod \min(x,y) \neq 0$ &

$\min(x,y) \bmod \big(\max(x,y) \bmod \min(x,y)\big) = 0$

example: a PROGRAM P and corresponding PROGRAM SCHEME S

Euclidean GCD program

precondition :

$x > 0$ and $y > 0$

1 : $m := \min(x, y)$

2 : $n := \max(x, y)$

3 : **while** $m \neq 0$

4 : $\quad r := (n \bmod m)$

5 : $\quad n := m$

6 : $\quad m := r$

7 : **return** n

corresponding program scheme

precondition :

$\mathbf{R}\,(x, \mathbf{c}\,) \wedge \mathbf{R}\,(y, \mathbf{c}\,)$

1 : $m := \mathbf{lo}\,(x, y)$

2 : $n := \mathbf{hi}\,(x, y)$

3 : **while** $\neg(m \doteq \mathbf{c}\,)$

4 : $\quad r := \mathbf{f}\,(n, m)$

5 : $\quad n := m$

6 : $\quad m := r$

7 : **return** n

Program execution is fully determined by the values of input variables x and y, *i.e.*, by constraints exclusively involving x and y and none of the program/internal variables $\{m, n, r\}$, *e.g.*, consider the number of times the loop body $\{4, 5, 6\}$ is executed.

For example, $\{4, 5, 6\}$ is executed twice iff:

$\min(x, y) \neq 0$ &	$\neg\left(\mathbf{lo}\; x\, y \doteq \mathbf{c}\,\right) \quad \wedge$
$\max(x, y)$ mod $\min(x, y) \neq 0$ &	$\neg\left(\mathbf{f}\,(\mathbf{hi}\; x\, y, \mathbf{lo}\; x\, y) \doteq \mathbf{c}\,\right) \quad \wedge$
$\min(x, y)$ mod $\left(\max(x, y) \text{ mod } \min(x, y)\right) = 0$	$\mathbf{f}\left(\mathbf{lo}\; x\, y, \mathbf{f}\,(\mathbf{hi}\; x\, y, \mathbf{lo}\; x\, y)\right) \doteq \mathbf{c}$

example: unwinding a PROGRAM SCHEME into an INFINITE FLOW DIAGRAM

precondition :

$\mathbf{R}(x, \mathbf{c}) \wedge \mathbf{R}(y, \mathbf{c})$

1 : $m := \mathbf{lo}(x, y)$

2 : $n := \mathbf{hi}(x, y)$

3 : **if** $\neg(m \doteq \mathbf{c})$ **then** $(7:$ **return** $n)$ **else**

4 : $r := \mathbf{f}(m, n)$

5 : $n := m$

6 : $m := r$

3 : **if** $\neg(m \doteq \mathbf{c})$ **then** $(7:$ **return** $n)$ **else**

4 : $r := \mathbf{f}(m, n)$

5 : $n := m$

6 : $m := r$

3 : **if** $\neg(m \doteq \mathbf{c})$ **then** $(7:$ **return** $n)$ **else**

4 : $r := \mathbf{f}(m, n)$

5 : $n := m$

6 : $m := r$

$\vdots \qquad \vdots$

example: unwinding a PROGRAM SCHEME into an INFINITE FLOW DIAGRAM

precondition :
$\mathbf{R}(x,\mathbf{c}) \wedge \mathbf{R}(y,\mathbf{c})$

1 : $m := \mathbf{lo}(x,y)$
2 : $n := \mathbf{hi}(x,y)$

3 : **if** $\neg(m \doteq \mathbf{c})$ **then** $(7:$ **return** $n)$ **else**
4 : $r := \mathbf{f}(m,n)$
5 : $n := m$
6 : $m := r$

3 : **if** $\neg(m \doteq \mathbf{c})$ **then** $(7:$ **return** $n)$ **else**
4 : $r := \mathbf{f}(m,n)$
5 : $n := m$
6 : $m := r$

3 : **if** $\neg(m \doteq \mathbf{c})$ **then** $(7:$ **return** $n)$ **else**
4 : $r := \mathbf{f}(m,n)$
5 : $n := m$
6 : $m := r$

$\vdots$ $\vdots$

- every diverging execution is described by an **infinite** sequence of instruction labels of the form:
  $1\ 2\ (3\ 4\ 5\ 6)^{\omega}$
- every converging execution is described by a **finite** sequence of instruction labels of the form:
  $1\ 2\ (3\ 4\ 5\ 6)^{*}\ 3\ 7$

example: unwinding a PROGRAM SCHEME into an INFINITE FLOW DIAGRAM

precondition :

$\mathbf{R}(x,\mathbf{c}) \wedge \mathbf{R}(y,\mathbf{c})$

1 : $m := \mathbf{lo}(x,y)$

2 : $n := \mathbf{hi}(x,y)$

3 : **if** $\neg(m \doteq \mathbf{c})$ **then** $(7:$ **return** $n)$ **else**

4 : $r := \mathbf{f}(m,n)$

5 : $n := m$

6 : $m := r$

3 : **if** $\neg(m \doteq \mathbf{c})$ **then** $(7:$ **return** $n)$ **else**

4 : $r := \mathbf{f}(m,n)$

5 : $n := m$

6 : $m := r$

3 : **if** $\neg(m \doteq \mathbf{c})$ **then** $(7:$ **return** $n)$ **else**

4 : $r := \mathbf{f}(m,n)$

5 : $n := m$

6 : $m := r$

$\vdots \qquad \vdots$

- every diverging execution is described by an **infinite** sequence of instruction labels of the form:
  $1\ 2\ (3\ 4\ 5\ 6)^\omega$
- every converging execution is described by a **finite** sequence of instruction labels of the form:
  $1\ 2\ (3\ 4\ 5\ 6)^*\ 3\ 7$
- every diverging execution is specified by an **infinite** set of quantifier-free first-order WFF's over the signature $\{\mathbf{R},\mathbf{lo},\mathbf{hi},\mathbf{f},\mathbf{c}\}$ and input variables $\{x,y\}$
- every converging execution is specified by a **finite** set of quantifier-free first-order WFF's over the signature $\{\mathbf{R},\mathbf{lo},\mathbf{hi},\mathbf{f},\mathbf{c}\}$ and input variables $\{x,y\}$

- Let P be a deterministic sequential program whose prim ops are the interpretations of the predicate symbols and function symbols of a signature Σ in a Σ-structure $\mathscr{M}$.
- Let $X \triangleq \{x_1, \ldots, x_m\}$, $Y \triangleq \{y_1, \ldots, y_n\}$, and $Z \triangleq \{z_1, \ldots, z_p\}$, be input variables, output variables, and program variables of P, with $m \geqslant 1$, $n \geqslant 0$, and $p \geqslant 0$.

  In particular, an execution of P is triggered by an assignment of values from the domains of $\mathscr{M}$ to the input variables X. If and when an execution of P terminates, the returned output is the set of values stored in the variables Y.
- Let S be the program scheme corresponding to program P, *i.e.*, the interpretation of S in $\mathscr{M}$, denoted $S^{\mathscr{M}}$, is exactly P.

- Let P be a deterministic sequential program whose prim ops are the interpretations of the predicate symbols and function symbols of a signature Σ in a Σ-structure $\mathscr{M}$.
- Let $X \triangleq \{x_1, \ldots, x_m\}$, $Y \triangleq \{y_1, \ldots, y_n\}$, and $Z \triangleq \{z_1, \ldots, z_p\}$, be input variables, output variables, and program variables of P, with $m \geqslant 1$, $n \geqslant 0$, and $p \geqslant 0$.

  In particular, an execution of P is triggered by an assignment of values from the domains of $\mathscr{M}$ to the input variables X. If and when an execution of P terminates, the returned output is the set of values stored in the variables Y.
- Let S be the program scheme corresponding to program P, *i.e.*, the interpretation of S in $\mathscr{M}$, denoted $S^{\mathscr{M}}$, is exactly P.
- **Theorem 1:** Let *Paths*$(S) \triangleq \{\pi_1, \pi_2, \ldots\}$ be the set of all finite execution paths in program scheme S. Let every test in S be a first-order WFF φ over signature Σ with FV$(\varphi) \subseteq X \cup Y \cup Z$.

  For every $\pi_i \in$ *Paths*(S) there is a first-order WFF α_i over Σ with FV$(\alpha_i) \subseteq \{x_1, \ldots, x_m\}$ such that for every execution of $P = S^{\mathscr{M}}$ on input values $\vec{a} \triangleq (a_1, \ldots, a_m)$:

  the execution converges by following path π_i iff $(\mathscr{M}, \vec{a}) \models \alpha_i$.
- Let *PathConstraints*$(S) \triangleq \{\alpha_1, \alpha_2, \ldots\}$ be the first-order WFF's thus defined over signature Σ with free variables in X.

- **Theorem 2** is a weaker version of **Theorem 1** that applies to common programming languages (Python, Java, Haskell, C, etc.) – why?

- **Theorem 2:** Let *Paths*$(S) \triangleq \{\pi_1, \pi_2, \ldots\}$ be the set of all finite execution paths in program scheme S. Let every test in S be a first-order literal (*i.e.*, an atomic or negated atomic WFF) over signature Σ with variables in $X \cup Y \cup Z$.

  For every $\pi_i \in$ *Paths*(S) there is a conjunction α_i of literals over Σ with variables in $\{x_1, \ldots, x_m\}$ such that for every execution of $P = S^{\mathscr{M}}$ on input values $\vec{a} \triangleq (a_1, \ldots, a_m)$:

  the execution converges by following path π_i iff $(\mathscr{M}, \vec{a}) \models \alpha_i$.

from PROGRAM SCHEMES to FIRST-ORDER LOGIC

- Let S be a program scheme whose prim ops are in the signature Σ and whose input variables are $X = \{x_1, \ldots, x_m\}$. Let $\mathscr{C}$ be a class of Σ-structures.
  Let $\Phi \triangleq \{\varphi_1, \varphi_2, \ldots\}$ be a set (possibly infinite) of first-order WFF's over signature Σ with $\mathrm{FV}(\varphi_i) \subseteq \{x_1, \ldots, x_m\}$ for every $i \geqslant 1$.

  We say that Φ *enforces totality* of program scheme S (*i.e.*, termination/convergence of all executions by S) in the class $\mathscr{C}$ iff:

  for every $\mathscr{M} \in \mathscr{C}$ and every m-tuple $\vec{a} \triangleq (a_1, \ldots, a_m)$ of inputs

  from the domains of $\mathscr{M}$, if $(\mathscr{M}, \vec{a}) \models \Phi$ then the execution of $S^{\mathscr{M}}(\vec{a})$ converges.

- Let S be a program scheme whose prim ops are in the signature Σ and whose input variables are $X = \{x_1, \ldots, x_m\}$. Let $\mathscr{C}$ be a class of Σ-structures.

  Let $\Phi \triangleq \{\varphi_1, \varphi_2, \ldots\}$ be a set (possibly infinite) of first-order WFF's over signature Σ with $\mathsf{FV}(\varphi_i) \subseteq \{x_1, \ldots, x_m\}$ for every $i \geqslant 1$.

  We say that Φ *enforces totality* of program scheme S (*i.e.*, termination/convergence of all executions by S) in the class $\mathscr{C}$ iff:

  for every $\mathscr{M} \in \mathscr{C}$ and every m-tuple $\vec{a} \triangleq (a_1, \ldots, a_m)$ of inputs

  from the domains of $\mathscr{M}$, if $(\mathscr{M}, \vec{a}) \models \Phi$ then the execution of $S^{\mathscr{M}}(\vec{a})$ converges.

- **Corollary:** The following are equivalent statements:

  1. $\Phi \triangleq \{\varphi_1, \varphi_2, \ldots\}$ enforces totality of program scheme S in class $\mathscr{C}$.
  2. For every $\mathscr{M} \in \mathscr{C}$ and all inputs $\vec{a} \triangleq (a_1, \ldots, a_m)$ from the domains of $\mathscr{M}$, it holds that if $(\mathscr{M}, \vec{a}) \models \Phi$ then $(\mathscr{M}, \vec{a}) \models \bigvee_{j \geqslant 1} \alpha_j$.
  3. For every $\mathscr{M} \in \mathscr{C}$ and all inputs $\vec{a} \triangleq (a_1, \ldots, a_m)$ from the domains of $\mathscr{M}$, it holds that $(\mathscr{M}, \vec{a}) \models \left(\bigwedge_{i \geqslant 1} \varphi_i \to \bigvee_{j \geqslant 1} \alpha_j \right)$.
  4. For every $\mathscr{M} \in \mathscr{C}$, it holds that $\mathscr{M} \models \forall \vec{x} \left(\bigwedge_{i \geqslant 1} \varphi_i \to \bigvee_{i \geqslant 1} \alpha_i \right)$.

  Note: If Φ is an infinite set, then $\bigwedge_{i \geqslant 1} \varphi_i$ is an *infinitary conjunction*, and thus ***not*** in the syntax of first-order logic. Likewise, $\bigvee_{i \geqslant 1} \alpha_i$ is an *infinitary disjunction*, and thus ***not*** in the syntax of first-order logic, when *PathConstraints*$(S) = \{\alpha_1, \alpha_2, \ldots\}$ is an infinite set.

HOW STRONG CAN WE HOPE TO MAKE THE PRECONDITIONS?

- We think of Φ as a set of *formal preconditions* for program scheme S.

  Question: Given an arbitrary program scheme S, can we formulate the preconditions Φ, as a set of first-order WFF's, to enforce totality of S?

HOW STRONG CAN WE HOPE TO MAKE THE PRECONDITIONS?

- We think of Φ as a set of *formal preconditions* for program scheme S.

  Question: Given an arbitrary program scheme S, can we formulate the preconditions Φ, as a set of first-order WFF's, to enforce totality of S?

- **Exercise:** Let S be an arbitrary program scheme over some signature Σ with input variables $X \triangleq \{x_1, \ldots, x_m\}$.

  Define an infinitary WFF Ψ (**note:** Ψ is not restricted to be first-order) over signature Σ with $\text{FV}(\Psi) \subseteq X$ such that for every Σ-structure $\mathscr{M}$ and all inputs $\vec{a} \triangleq (a_1, \ldots, a_m)$ from the domains of $\mathscr{M}$, it holds that

  if $(\mathscr{M}, \vec{a}) \models \Psi$ then the execution of $S^{\mathscr{M}}(\vec{a})$ converges.

  In words, Ψ enforces totality of S in all Σ-structures $\mathscr{M}$, not restricted to any particular class.

THE UNWIND PROPERTY

- Let S be a program scheme over some signature Σ with input variables $X \triangleq \{x_1, \ldots, x_m\}$.

  We say S *unwinds* in a class $\mathscr{C}$ of Σ-structures iff there is a finite subset $\{\pi_1, \ldots, \pi_k\} \subseteq \mathit{Paths}(S)$ and corresponding finite subset $\{\alpha_1, \ldots, \alpha_k\} \subseteq \mathit{PathConstraints}(S)$ such that, for all $\mathscr{M} \in \mathscr{C}$ and all inputs $\vec{a} \triangleq (a_1, \ldots, a_m)$ from the domains of $\mathscr{M}$:

  the execution of $S^{\mathscr{M}}(\vec{a})$ converges iff $(\mathscr{M}, \vec{a}) \models \alpha_1 \vee \ldots \vee \alpha_k$.

  Informally, only a finite set of $k \geqslant 1$ paths are used by converging executions of S. Put differently, if S unwinds in the class $\mathscr{C}$, then S is equivalent to a “trivial” (*i.e.*, loop-free) program scheme.

---

[1] Strictly, $\{\mathscr{M} \mid \mathscr{M} \models \Phi\}$ is the class defined as $\{\mathscr{M} \mid (\mathscr{M}, \vec{a}) \models \Phi$ for all m-tuples $\vec{a}$ from the domains of $\mathscr{M}\}$. FV$(\Phi) \subseteq \{x_1, \ldots, x_m\}$ and $\vec{a}$ is an assignment of values to the free variables in Φ.

THE UNWIND PROPERTY

- Let S be a program scheme over some signature Σ with input variables $X \triangleq \{x_1,\ldots,x_m\}$.

  We say S *unwinds* in a class $\mathscr{C}$ of Σ-structures iff there is a finite subset $\{\pi_1,\ldots,\pi_k\} \subseteq$ *Paths*(S) and corresponding finite subset $\{\alpha_1,\ldots,\alpha_k\} \subseteq$ *PathConstraints*(S) such that, for all $\mathscr{M} \in \mathscr{C}$ and all inputs $\vec{a} \triangleq (a_1,\ldots,a_m)$ from the domains of $\mathscr{M}$:

  the execution of $S^{\mathscr{M}}(\vec{a})$ converges iff $(\mathscr{M},\vec{a}) \models \alpha_1 \vee \ldots \vee \alpha_k$.

  Informally, only a finite set of $k \geqslant 1$ paths are used by converging executions of S. Put differently, if S unwinds in the class $\mathscr{C}$, then S is equivalent to a "trivial" (*i.e.*, loop-free) program scheme.

- **Theorem 3:** Let S be a program scheme over signature Σ with input variables $X \triangleq \{x_1,\ldots,x_m\}$. Let Φ be a set (possibly infinite) of first-order WFF's over signature Σ with FV$(\Phi) \subseteq X$ and let $\mathscr{C} \triangleq \{\, \mathscr{M} \mid \mathscr{M} \models \Phi \,\}$.[1]

  If Φ enforces totality of S in the class $\mathscr{C}$, then S unwinds in the class $\mathscr{C}$.

  In other words, we cannot constrain the interpretations in $\mathscr{C}$ for a program scheme S by first-order conditions Φ in order to ensure termination – unless we also make superfluous the presence of the loops in S.

---

[1] Strictly, $\{\, \mathscr{M} \mid \mathscr{M} \models \Phi \,\}$ is the class defined as $\{\, \mathscr{M} \mid (\mathscr{M},\vec{a}) \models \Phi$ for all m-tuples $\vec{a}$ from the domains of $\mathscr{M} \,\}$. FV$(\Phi) \subseteq \{x_1,\ldots,x_m\}$ and $\vec{a}$ is an assignment of values to the free variables in Φ.

- **Proof Sketch for Theorem 3**: By contradiction. Assume that Φ enforces totality of S in the class $\mathscr{C}$, but yet S does not unwind in the class $\mathscr{C}$, and we then get a contradiction.

  Consider *PathConstraints*$(S) = \{\alpha_1, \alpha_2, \dots\}$. By the Corollary of Theorems 1 and 2 (see part 2 in particular), together with the preceding assumption, we must have: For every $k \geqslant 1$ there is a Σ-structure $\mathscr{M} \in \mathscr{C}$ and there are inputs $\vec{a} \triangleq (a_1, \dots, a_m)$ from the domains of $\mathscr{M}$ such that $(\mathscr{M}, \vec{a}) \models \Phi \cup \{\neg\alpha_1, \dots, \neg\alpha_k\}$ (straightforward details of this argument are omitted).

  Hence, for every $k \geqslant 1$, the set of first-order WFF's $\Phi \cup \{\neg\alpha_1, \dots, \neg\alpha_k\}$ is consistent. Hence, by Compactness of first-order logic, the full set $\Phi \cup \{\neg\alpha_1, \neg\alpha_2, \dots\}$ is consistent/satisfiable. Hence, there is a Σ-structure $\mathscr{M} \in \mathscr{C}$ and there are inputs $\vec{a} \triangleq (a_1, \dots, a_m)$ from the domains of $\mathscr{M}$ such that $(\mathscr{M}, \vec{a}) \models \Phi \cup \{\neg\alpha_1, \neg\alpha_2, \dots\}$ and, in particular, $(\mathscr{M}, \vec{a}) \models \{\neg\alpha_1, \neg\alpha_2, \dots\}$ which implies that $S^{\mathscr{M}}(\vec{a})$ does not converge. But this contradicts the assumption that Φ enforces totality of S in the class $\mathscr{C}$ (again, straightforward details are omitted).

(THIS PAGE INTENTIONALLY LEFT BLANK)