

گزارش نهایی پروژه درس «روش‌های صوری»

Detecting Redundant Preconditions

خلاصه، پیاده‌سازی و شبیه‌سازی

مهندی مالوردی*

چکیده

این گزارش مقاله‌ی Detecting Redundant Preconditions را خلاصه می‌کند و یک پیاده‌سازی/شبیه‌سازی آموزشی برای بررسی افزونگی پیش‌شرط‌ها در یک دامنه محدود ارائه می‌دهد. هدف، مشاهده‌ی تفاوت روش‌های مبتنی بر Dependency و Implication با روش «حذف پیش‌شرط و بررسی دوباره» در سناریوهای ساده است.

کلمات کلیدی: قراردادها، پیش‌شرط، افزونگی، Design by Contract، روش‌های صوری

۱ خلاصه مقاله مرجع

عنوان: Detecting Redundant Preconditions
نویسنده‌ان: Nicola Thoben, Heike Wehrheim
منبع: FormaliSE 2025 (IEEE/ACM)
DOI: 10.1109/FORMALISE66629.2025.00015

۱.۱ هدف و دامنه

مقاله روی «کیفیت قراردادها» تمرکز دارد، نه روی تولید قرارداد (contract synthesis) و نه روی اعتبارسنجی صرف. فرض مقاله این است که برنامه‌ها نسبت به قرارداد فعلی صحیح هستند و

Student number: 404443150*

سؤال اصلی این است که آیا بخشی از پیششرط‌ها واقعاً برای برقرار شدن پس‌شرط لازم هستند یا خیر. نکته‌ی مهم مقاله این است که در برخی سناریوها—اگر یک پیششرط از نظر صحیح پس‌شرط «لازم» نباشد—ممکن است از نظر مستندسازی یا تعریف دامنه‌ی ورودی‌های معتبر، نگه داشتن آن مطلوب باشد؛ با این حال کشف افزونگی می‌تواند قرارداد را ساده‌تر، خواناتر و قابل استفاده‌تر کند.

۲.۱ پیشزمینه مفهومی

قراردادها در **Design by Contract** قرارداد تابع/ماژول معمولاً با Preconditions و Postconditions بیان می‌شود. در مدل مقاله، پیششرط‌ها و پس‌شرط‌ها به ترتیب به صورت assert و assume در ابتدای/انتهای برنامه (یا تابع بسته) قرار داده می‌شوند تا ابزارهای تحلیل/اثبات بتوانند آن‌ها را بررسی کنند.

نمودار جریان کنترل و تحلیل گزاره‌ای برای تحلیل، مقاله از مفاهیم رایج در راستی‌آزمایی نرم‌افزار استفاده می‌کند: نمایش برنامه به صورت CFA (نمودار جریان کنترل)، توسعه‌ی آن با یال‌های predi-assume/assert و مدل کردن نقضی assert با رسیدن به مکان خطأ. همچنین از CEGAR و ARG (پالایش مبتنی بر ضدنمونه) برای ساخت cate analysis (گراف دسترسی‌پذیری انتزاعی) استفاده می‌شود.

۳.۱ تعریف رسمی افزونگی

مقاله افزونگی را بر اساس «درستی برنامه نسبت به قرارداد» تعریف می‌کند:

- **افزونگی تکی:** پیششرط pre_i در مجموعه‌ی Pre زائد است اگر با حذف آن همچنان برنامه پس‌شرط‌ها را برقرار کند.
- **افزونگی گروهی:** یک مجموعه از پیششرط‌ها وقتی زائد گروهی است که بتوان همه‌ی آن‌ها را همزمان حذف کرد و قرارداد همچنان برقرار بماند.

این دو مفهوم مهم‌اند چون ممکن است هر پیششرط به تنها‌ی «قابل حذف» باشد، ولی حذف همزمان چند پیششرط باعث شکستن قرارداد شود (وابستگی بین پیششرط‌ها/نقش‌شان در محدود کردن دامنه‌ی حالات اولیه).

۴.۱ رویکرد ساده اما پرهزینه (Baseline)

تعریف افزونگی مستقیماً یک روش ساده پیشنهاد می‌کند: برای هر پیششرط (یا مجموعه‌ای از آن‌ها) یک بار آن را حذف کنیم و دوباره با یک راستی‌آزما بررسی کنیم که آیا پس‌شرط‌ها هنوز اثبات

می‌شوند یا نه. مشکل: اجرای چندباره‌ی ابزارهای اثبات/راستی‌آزمایی برای هر برنامه بسیار پرهزینه است، بنابراین مقاله دنبال تکنیک‌های کاراتر است.

۱.۵. تکنیک‌های پیشنهادی برای کشف افزونگی

مقاله سه تکنیک با توازن متفاوت بین دقت و هزینه ارائه می‌دهد:

(۱) **IC – بررسی استنتاج منطقی (Implication Checking)** در این روش خود پیش‌شرط‌ها با هم مقایسه می‌شوند و برنامه/پس‌شرط به صورت مستقیم وارد تحلیل نمی‌شود. ایده: اگر یک پیش‌شرط از ترکیب سایر پیش‌شرط‌ها نتیجه شود، در حضور آن‌ها اضافه است. مقاله همچنین توضیح می‌دهد که افزونگی تکی لزوماً به افزونگی گروهی تبدیل نمی‌شود؛ برای همین یک رویه‌ی تکراری برای بررسی حذف گروهی پیش‌شرط‌های «قابل نتیجه‌گیری» ارائه می‌کند.

(۲) **DC – تحلیل وابستگی (Dependency Calculation)** در این روش، برنامه به صورت CFA توسعه‌یافته (با یال‌های assume/assert و مکان خطای) مدل می‌شود و سپس وابستگی‌های داده/کنترل بین پیش‌شرط‌ها و پس‌شرط‌ها بررسی می‌شود. اگر یک پیش‌شرط به هیچ پس‌شرطی (از منظر وابستگی نحوی) وابسته نباشد، می‌توان آن را زائد فرض کرد. این روش سریع است، اما چون صرفاً وابستگی‌های نحوی را می‌بیند، ممکن است پیش‌شرطی که وابستگی نحوی دارد ولی از نظر معنایی لازم نیست را زائد تشخیص ندهد.

(۳) **PC – تحلیل گزاره‌ای/انتزاعی (Predicate Checking)** این روش یک بار تحلیل گزاره‌ای را اجرا می‌کند تا ARG ساخته شود. ایده‌ی کلیدی: اگر در یال متناظر با یک پیش‌شرط (assume(b))، گزاره‌ی انتزاعی قبل و بعد از آن تغییری نکند (یعنی اضافه شدن آن پیش‌شرط در محاسبه‌ی انتزاع اثر نگذارد)، آن پیش‌شرط در این اثبات «لازم» نبوده و می‌تواند زائد باشد. مقاله برای نسخه‌ی مبتنی بر ARG یک قضیه‌ی صحت (soundness) بیان می‌کند: اگر عبور از یال موجب تقویت predicate نشود، جایگزینی آن با assume(true) همچنان شواهد کافی برای صحت قرارداد را حفظ می‌کند.

۱.۶. ارزیابی تجربی مقاله

داده‌ها و بنچمارک برای ارزیابی، نویسنده‌گان مجموعه‌ای از ۶۲ برنامه C را انتخاب می‌کنند (از منابع مختلف) و ابتدا با CPAchecker بررسی می‌کنند که نسبت به قراردادهای موجود صحیح هستند. سپس با سه نوع تبدیل، پیش‌شرط‌های زائد تزریق می‌کنند:

• **(Independency): ۱ Type** معرفی متغیر/قید اضافی که در صحت پس‌شرط نقشی ندارد.

۲ Type (Implication): ساخت قیدی که توسط یک پیششرط لازم دیگر نتیجه می‌شود.
با کمک حل‌کننده‌ی (Eldarica).

۳ Type (Range): افروden کردن «جهت مخالف» برای متغیری که یک کران لازم دارد (مثلًاً اگر $0 < x \leq 100$ است، x اضافه می‌شود) تا قید بازه‌ای بسازد که از نظر صحت پس‌شرط زائد است.

فرضیه‌ها

مقاله سه فرضیه مطرح می‌کند:

- ۱H (اثربخشی): IC و DC قابل مقایسه‌ی ساده نیستند و PC باید از هر دو بهتر باشد.
- ۲H (کامل بودن): انتظار می‌رود PC (در صورت اتمام CEGAR) کامل باشد، اما نتایج تجربی خلاف این را نشان می‌دهد.
- ۳H (کارایی): IC/DC بسیار سبک‌تر از PC هستند.

نتیجه‌های اصلی در آزمایش‌های مقاله، PC بیشترین تعداد پیششرط‌های زائد را تشخیص می‌دهد (به خصوص در نوع Range) ولی از نظر کامل بودن در پیاده‌سازی CPAchecker بی‌نقص نیست. همچنین PC از نظر زمان/حافظه پرهزینه‌تر از روش‌های مبتنی بر CFA است، در حالی که IC و DC بسیار سریع‌ترند.

۷.۱ جمع‌بندی و کارهای آینده (طبق مقاله)

مقاله سه تکنیک عملی برای کشف افزونگی پیششرط‌ها ارائه می‌دهد و نشان می‌دهد بین سرعت و دقیقت یک trade-off جدی وجود دارد. به عنوان کار آینده، توسعه‌ی روش‌هایی برای کشف «روابط علی» بین پیششرط‌ها و پس‌شرط‌ها و همچنین استفاده از ACSL برای نوشتن قراردادها پیشنهاد می‌شود. در بسته‌ی پروژه، فایل PDF مقاله مرجع نیز پیوست شده است تا خواننده در صورت نیاز به جزئیات کامل، مستقیماً به متن اصلی مراجعه کند.

۸.۱ مسئله

در «Design Contract by Design» قراردادها معمولاً با مجموعه‌ای از پیششرط‌ها (Preconditions) و پس‌شرط‌ها (Postconditions) نوشته می‌شوند. هدف مقاله بررسی کیفیت قراردادهاست: آیا برخی پیششرط‌ها زائد هستند (یعنی حتی بدون آن‌ها هم پس‌شرط‌ها برقرار می‌مانند)؟

۹.۱ تعریف افزونگی (Redundancy)

یک پیششرط i در مجموعه Pre زائد است اگر برنامه با حذف آن پیششرط، همچنان قرارداد را ارضاء کند (به صورت $(\text{Pre} \setminus \{\text{pre}_i\}, \text{Post}) = P$). مقاله علاوه بر افزونگی تکی، مفهوم افزونگی گروهی (group-redundant) را هم تعریف می‌کند.

۱۰.۱ روش‌های پیشنهادی مقاله

مقاله سه تکنیک برای کشف پیششرط‌های زائد ارائه می‌کند:

۱. **Implication Checking (IC)**: بررسی می‌کند آیا pre_i به صورت منطقی از بقیه‌ی پیششرط‌ها نتیجه می‌شود یا نه.

۲. **Dependency Calculation (DC)**: روی CFA توسعه یافته، وابستگی‌های داده/کنترل بین `assume` و `assert` را بررسی می‌کند؛ اگر پیششرط به هیچ پسشرطی وابسته نباشد، زائد است (روش نحوی/سینتکتیک).

۳. **Predicate Checking (PC)**: با CEGAR و predicate analysis و ARG ساخت بررسی می‌کند آیا در یال (b) `assume` بعد از آن تغییر/تقویت می‌شود یا نه؛ اگر تقویت نشود، آن پیششرط زائد است.

۱۱.۱ نتایج تجربی مقاله (خیلی خلاصه)

روی ۶۲ برنامه C و ۱۵۵ پیششرط زائد تزریق شده:

IC: 61/155 •

DC: 62/155 •

PC: 147/155 •

همچنین PC منابع بیشتری (زمان/حافظه) مصرف می‌کند.

۱۲ نوآوری این پروژه (افزونگی گروهی با شاهد نقض)

- ایده/نوآوری: علاوه بر تشخیص افزونگی تکی، یک ماژول افزونگی گروهی اضافه شده است که وقتی مجموعه‌ای از پیششرط‌ها به صورت تکی زائد هستند اما حذف همزمان آن‌ها قرارداد را می‌شکند، یک شاهد نقض (ورودی مشخص) تولید می‌کند.

- چرایی ارزشمند بودن: این کار، خروجی را از یک برچسب ساده‌ی «زائد/غیرزائد» به یک نتیجه‌ی explainable تبدیل می‌کند و برای ارائه/دیباگ قراردادها بسیار قابل استفاده است.
- جزئیات پیاده‌سازی: ابزار این موارد را محاسبه می‌کند:
 - مجموعه‌ی پیش‌شرط‌های single-redundant
 - یک مجموعه‌ی group-redundant (با حذف حریصانه)،
 - و در صورت عدم افزونگی گروهی، .input counterexample
- مثال ارائه‌ای (Counterexample): در فایل examples/group_redundancy.js هر سه پیش‌شرط به صورت تکی زائد هستند، اما حذف هم‌زمان همه‌ی آن‌ها قرارداد را می‌شکند و ابزار یک شاهد نقض مانند $a=-2, b=-2$ تولید می‌کند.
خروجی: .outputs/group_redundancy_report.json

(خلاصه) خروجی نمونه‌ی

```
single_redundant_indices: [0, 1, 2]
all_single_is_group_redundant: false
counterexample_if_not_group: {"a": -2, "b": -2}
```

۳ پیاده‌سازی و نتایج اجرای کد

۱.۳ توضیح پیاده‌سازی

- فایل fm_project/redundancy_checker.py یک ابزار ساده است که:
- یک برنامه‌ی کوچک را از روی فایل JSON می‌خواند (شامل post, pre و بدنی برنامه با (assign/while/if).
 - ورودی‌ها را در یک بازه‌ی محدود (bounded domain) شمارش (enumerate) می‌کند.
 - صحت قرارداد را در این دامنه بررسی می‌کند.
 - برای هر پیش‌شرط، با حذف آن و اجرای دوباره، افزونگی تکی را بررسی می‌کند.
 - یک بررسی IC-like هم انجام می‌دهد: آیا پیش‌شرط از بقیه‌ی پیش‌شرط‌ها (در همان دامنه محدود) نتیجه می‌شود یا نه.

۲.۳ نتایج نمونه (Example)

نمونه‌ی examples/sub.json نسخه‌ی ساده‌شده‌ی مثال مقاله/اسلاید است (متغیرهای N و M و حلقه). دستور اجرا:

```
.venv/bin/python main.py --spec examples/sub.json
```

خروجی اجرا: خروجی دقیق اجرا در فایل outputs/run_results.txt ذخیره شده است. خلاصه‌ی نتیجه روی این مثال (در دامنه محدود تعریف شده در JSON):

- پیش‌شرط ضروری: $N \geq 0$
- پیش‌شرط‌های زائد: $M \geq 0, N \geq -100, N \leq 1000$
- نکته: در بررسی IC-like، پیش‌شرط $0 \leq M$ نتیجه نمی‌شود اما با حذف همچنان قرارداد برقرار است؛ این دقیقاً نمونه‌ای از تفاوت «استنتاج از پیش‌شرط‌ها» با «لازم بودن برای برنامه» است.

۳.۳ محدودیت‌ها

این پیاده‌سازی یک آموزشی است و به جای تحلیل کامل C و ابزارهایی مثل CPAchecker، از bounded execution استفاده می‌کند؛ بنابراین نتیجه‌ها «تضمين کامل برای همه‌ی ورودی‌ها» نیست مگر دامنه ورودی‌ها کامل باشد.

۴ شبیه‌سازی (Benchmark)

برای مقایسه‌ی ایده‌های مقاله به صورت آموزشی، یک شبیه‌سازی کوچک روی مجموعه‌ای از برنامه‌های مصنوعی انجام شد. در هر برنامه یک پیش‌شرط ضروری و سه پیش‌شرط زائد از سه نوع Range و Implication، Independency شدند:

- IC-like: بررسی استنتاج منطقی از سایر پیش‌شرط‌ها
- DC-like: تحلیل وابستگی نحوی از متغیرهای پس‌شرط
- VC: حذف پیش‌شرط و بررسی مجدد قرارداد (روش معنایی در دامنه محدود)

این شبیه‌سازی با دستور زیر اجرا شد و خروجی در outputs/simulation_summary.txt ذخیره گردید:

نوع پیششرط زائد	تعداد واقعی	IC-like	DC-like	VC
Independency	۳۹	۰	۳۹	۳۹
Implication	۳۹	۳۹	۰	۳۹
Range	۳۹	۰	۰	۳۹
Total	۱۱۷	۳۹	۳۹	۱۱۷

```
.venv/bin/python main.py --simulate --sim_n=39 --sim_seed=1
```

نتیجه‌ی شبیه‌سازی (۳۹ برنامه، مجموع ۱۱۷ پیششرط زائد):

۵ فرآیند اجرای پروژه

۱. انتخاب مقاله مرجع و مطالعه (PDF) مقاله + اسلاید).
۲. استخراج تعاریف کلیدی (افزونگی تکی/گروهی، ایده‌ی روش‌های IC/DC/PC).
۳. پیاده‌سازی یک prototype کوچک برای بازتولید ایده‌ی «حذف پیششرط و بررسی دوباره» در قالب bounded checking.
۴. اجرای ابزار روی مثال‌ها و ثبت نتایج.
۵. آماده‌سازی گزارش و بسته‌ی نهایی زیپ.

۶ کدهای پروژه / گیت‌هاب

- کد پروژه کوچک است و به صورت فایل‌های همین زیپ پیوست می‌شود.
- اگر لازم شد روی گیت‌هاب آپلود شود: (لینک اینجا قرار گیرد)

۷ پیوست‌ها

- مقاله مرجع (PDF) و اسلایدها در فایل زیپ نهایی قرار داده می‌شوند.

منابع

N. Thoben, H. Wehrheim, "Detecting Redundant Preconditions," FormaliSE 2025, DOI: 10.1109/FORMALISE66629.2025.00015.