FUTÁSI IDEJŰ ELLENŐRZÉS

Fülöp István Marcell

2011.

BEVEZETÉS

A futási idejű ellenőrzés viszonylag új tudományterület, melynek alapja, hogy a rendszer futását megfigyeljük, elemezzük abból a szempontból, hogy a pontosan megfogalmazott tulajdonságok hogyan teljesülnek. Ez a dinamikus ellenőrzés sok olyan esetben is alkalmazható, amikor a statikus, tervezési idejű ellenőrzés nem, mert minden ellenőrzés ellenére adódhatnak problémák a futtatás közben. A pontosan megfogalmazott felügyelet és ellenőrzés az alábbi célokra használható:

- a program megértése
- a rendszer használatának megértése
- a biztonsági stratégia felügyelete
- hibakeresés, tesztelés
- ellenőrzés, helyességbizonyítás
- hibavédelem
- a viselkedés megváltoztatása (pl. helyreállítás)

A futó rendszert úgy tekinthetjük, mint állapotokból és eseményekből álló, ún. futási nyomvonalak létrehozóját. Ezek a nyomvonalak sokféleképpen feldolgozhatók:

- formalizált tulajdonságok ellenőrzése
- viselkedési modellek szimulációjának irányítása
- elemzés speciális algoritmusokkal
- megjelenítés

- statisztikus elemzés
- rendszertulajdonságok gépi tanulása

A futási idejű ellenőrzés kutatása két nagy területet fed le. Az egyik a nyomvonalak ellenőrzésének területe, ide tartoznak a tulajdonság-specifikációk és az azok adott nyomvonalon való ellenőrzésére szolgáló algoritmusok. A másik a nyomvonalak előállításának területe, ide tartozik a megfigyelések módja és azok rögzítése.

FELMŰSZEREZÉS

A futási idejű ellenőrző rendszerek általában valamilyen felműszerezést használnak a megfigyelések kinyerése érdekében. Természetesen ez a két terület nem független. Egy nyomvonal-előállító eljárásnak biztosítania kell, hogy egy tulajdonság ellenőrzéséhez minden szükséges megfigyelés rögzítésre kerüljön, ugyanakkor az irreleváns megfigyelések előállítása növeli az ellenőrzési költséget. A nyomvonal-előállítás kutatásának tárgya az, hogy egy adott tulajdonsághoz pontosan a megfigyelések megfelelő halmazát rögzítsük. Ez az összefüggés megfordítható, ha a nyomvonal-előállítás eljárása rögzített, fel lehet tenni a kérdést, hogy mely tulajdonság-specifikációs nyelv lenne a megfelelő.

A legtöbb eszköz aktív műszerezést használ, azaz kódszondák beillesztését a futó rendszerbe. A kevés kivételek egyike a BusMOP, a MOP keretrendszer megtestesülése PCI buszok felügyeletére. A BusMOP passzív műszerezést használ, közvetlenül a buszon lévő forgalmat figyeli meg. Hogy elkerüljük a hiányzó releváns megfigyeléseket, és lehetővé tegyük a futási idejű ellenőrzés nagy rendszerekre való skálázását, a műszerezésnek automatikusnak kell lennie. Az automatikus műszerezéshez szükséges a tárgyrendszer elemzése, hogy azonosítsuk, hol történnek az érdekes események.

A forráskód műszerezése megvalósításának egyre népszerűbb útja az aspektus-orientált programozáson keresztül vezet. Az aspektus-orientált technikák a futási idejű ellenőrző rendszerekben a J-Lo és a Hawk eszközökben mutatkoztak be. Még szorosabb kapcsolat jött létre a futási idejű ellenőrzés és az aspektus-orientált felműszerezés között az AspectBenc fordítóban a nyomillesztés elképzelését felhasználva. A nyomillesztés az AspectJ nyelv egy kiterjesztése, mely események reguláris mintáját fogja el. A nyomillesztéshez azonban aspektus-szemantika is tartozik, így a fordító által automatikusan szőhető. Azóta számos futási idejű ellenőrző eszköz, ideértve a MOP-ot és a Larva-t, az AspectJ-t használja a felműszerezéshez. Más programnyelvekhez az aspektus-orientált műszerezés kevésbé gyakori. Itt megemlítjük az InterAspect rendszert, amely a GCC fordító köztes reprezentációjának alapján végez felműszerezést, így a GCC külső felületű nyelvekhez biztosít aspektus-orientált felműszerezést, különösen a C-hez és a C++-hoz.

A felműszerezett rendszer és a felügyelet közötti interakció módja alapján megkülönböztetünk szinkron és aszinkron felügyeletet. A szinkron felügyelet esetén, ha a rendszer egy releváns megfigyelést állít elő, a további végrehajtás megáll, amíg a felügyelet megerősíti, hogy nem történt szabálytalanság. A szinkron felügyelet a bizonyosságnak az aszinkron rendszernél magasabb fokát nyújthatja, mert blokkolhat egy veszélyes lépést.

A FUTÁSI IDEJŰ ELLENŐRZÉS TÍPUSAI

A futási idejű ellenőrzésnek a tulajdonság-specifikáció alapján három nagy típusát különböztethetjük meg. Az első típusban a tulajdonságok a végrehajtási történetre vonatkoznak, kiértékelésük pedig a nyomvonal állapotai alapján történik. Ez a típus a statikus modellellenőrzés közvetlen kiterjesztése, az algoritmusok a rendszer egy modelljét ellenőrzik egy temporális logikában vagy hasonló formalizmusban – pl. reguláris kifejezésekkel és állapotgépekkel – megfogalmazott tulajdonság szerint. Ahogy a nyomvonal alakul, az ellenőrzésnek minden egyes új megfigyelés esetén a nyomvonal elejétől történő megismétlése fölösleges, on-line vagy inkrementális algoritmusok szükségesek. Ezek legszéleskörűbb eszközkészlete a MOP keretrendszer, amely támogatja a specifikációs formalizmusok nagy választékát, pl. a temporális logikákat és a véges automatákat, valamint számos megfigyelési tárgyat, pl. Java-programokat és buszmegfigyelést.

Különböző specifikációs formalizmusokhoz való futási idejű ellenőrzési algoritmusok összehasonlítása azt mutatja, hogy sok közös van bennük. Mindegyik algoritmus rendelkezik egy ellenőrző állapottal, amely frissül, ha a tárgyrendszerből új megfigyelés érkezik. Ez a látszólag különböző formalizmusú algoritmusok közötti hasonlóság vezetett az egyedi felügyelő logikák kutatásához, amelyek felhasználhatók közös alapformalizmusként felügyelők specifikálásához.

Az Eagle explicit fixpont-operátorokkal rendelkező logika, amely alkalmas jövő- és múlt idejű temporális logikák, intervallum logikák, kiterjesztett reguláris kifejezések és egyéb formalizmusok megvalósítására. Egy Eagle-specifikációkhoz való futtatómotor rugalmas rendszerré teszi azt ellenőrzők megvalósításához. A RuleR ezt a megközelítést egy lépéssel továbbviszi, amikor az explicit fixpont-operátorokat a megfigyelések hatására egymást aktiváló szabályokkal helyettesíti. A Ruler alacsonyabb szintű logika, melyben egy általában használt logika magas szintű szemantikájának leírása több erőfeszítést igényel. Emellett azonban az ellenőrző állapot sokkal finomabb kézbentartását teszi lehetővé, hatékonyabb ellenőrzőkhöz vezetve, melyek könnyebben megvalósítható és karbantartható rendszert eredményeznek.

A második típusban a tulajdonságok a rendszer moduljai közötti, előfeltétel-utófeltétel felépítésű szerződések. Ezek a tulajdonságok általában egy állapotot vagy egy egymásra következő állapotok közötti változást írnak le. A szerződések jellemzően állapot-kijelentések, azaz a pillanatnyi állapotbeli rendszerváltozókra vonatkozó kifejezések. Mint más helyességi tulajdonságokat, a szerződéseket is statikusan lehet ellenőrizni, de gyakran futási időben is ellenőrzik őket váratlan szabálytalanságok tekintetében. A korábban tárgyalt temporális specifikációktól eltérően a szerződés-ellenőrzés jellemzően szorosabb egységet alkot a rendszer végrehajtásával, és fogalmilag a rendszer részének tekinthető. A szerződések szemantikája meghatározza, hogy a szerződéseket mely pontokon kell ellenőrizni.

Néhány nyelv, mint az Eiffel és a Spec#, külön szintaxist biztosítanak a szerződésekhez. Más nyelvi keretrendszerek támogatják a szerződések kódként történő beágyazását, általában a szerződésekhez szükséges alapkönyvtár által. Sok megjegyzés-alapú szerződés-keretrendszer van Java-hoz, mint a JML és a Jass. Mindkettő speciálisan felismerhető megjegyzésként írt specifikációkon alapul, melyek később eszközökkel kinyerhetők. Néhány szerződés-ellenőrző rendszer, többek között a Jass, kiterjeszti a szabványos szerződéseket nyomvonali kijelentésekkel, melyekkel korlátozásokat lehet specifikálni az osztály eljáráshívásainak

sorozatára. Ezzel a kiterjesztéssel a szerződések és a temporális tulajdonságok közötti különbség kezd eltűnni.

A harmadik típusban a tulajdonságok megsértését a specifikáció-nélküli felügyelet ellenőrző algoritmusai észlelik. Ezek a tulajdonságok általában konkurrens végrehajtáshoz kötődnek, mint pl. versenyhelyzet-mentesség, atomitás, sorosíthatóság, stb.

KÖLTSÉGCSÖKKENTÉS

A futási idejű ellenőrzés minden területén fontos a költségek kézbentartása. A költségek jelentős részét a megfigyelések kinyeréséhez szükséges felműszerezés teszi ki. Sok rendszerváltozót tartalmazó bonyolult tulajdonságok futási idejű ellenőrzése nagy költséget jelent a rendszer teljesítményének kárára. Sok munkát fektettek az ellenőrzési költség csökkentésébe a statikus elemzési technikák és az utólagos futási idejű ellenőrzés ötvözése által.

A statikus és dinamikus analízis ötvözésének ötlete a típusállapot-analízis területéről származik. A típusállapot-tulajdonságok szálak tulajdonságai egy programban, és hasonlóak a futási idejű ellenőrzés irodalmából ismert viselkedési specifikációkhoz. Egy jellemző típusállapot-tulajdonságot egy állapotgépként lehet felfogni, amely egy programban az érvényes könyvtárhívások sorozatait korlátozza. A maradék-analízis csökkentheti azt az állapotgépet, amelyet futási időben analizálni kell. A maradék-analízist kiegészíti az a módszer, ahol a felügyelni kívánt kód statikus analízisét azzal a céllal hajtják végre, hogy azonosítsák azokat a műszerezési pontokat, melyek biztonságosan eltávolíthatók. Ennek az eredménye megvalósításra került a Clara eszközben. Néhány esetben ugyanakkor elfogadható a felügyelet pontosságának csökkentése néhány végrehajtási esemény elhagyása által. Ezekben az esetekben kompromisszum van a pontosság és az alacsonyabb költség között.

VISSZACSATOLÁS ÉS FUTÁSI IDEJŰ KÉNYSZERÍTÉS

Végül, egy futási idejű ellenőrzési keretrendszernek megfelelő visszajelzést kell szolgáltatnia a felhasználónak. Egy futó rendszer esetén az ellenőrzésből származó információ segíthet egy problémából való talpraállásban. Hogy milyen a helyes visszajelzés, és milyen hatásai lehetnek a rendszer működésére, az a futási idejű ellenőrzés kutatásának egy másik fontos iránya.

Egy futási idejű ellenőrző rendszer tervezésénél fontos kérdés, hogy mi történjen, ha egy szabálytalanságot észlelünk. Ha a rendszer tesztelés alatt áll, elég lehet értesíteni az operátort, aki megállítja a rendszert, és megállapítja a problémát. Ugyanakkor annak érdekében, hogy a futási idejű ellenőrzésnek a tervezési idejű verifikációval szembeni, telepítés utáni alternatívája elképzelését megvalósítsuk, ennél több beavatkozás szükséges. Számos valós idejű ellenőrző rendszer, többek között a MaC és a MOP, rendelkezik a képességgel, hogy felhasználó által meghatározott talpraállító eljárásokat hívjon meg. A felügyelet specifikációi lehetővé teszik a felhasználónak, hogy meghatározza, milyen eljárások hívódjanak meg, ha egy adott szabálytalanság történik, és milyen információ legyen átadva a talpraállító eljárásnak. A rendszer

részlegesen reset-elődhet vagy hibamentő módba kapcsolhat. Néhány helyzetben, pl. teljesítmény és szolgáltatásminőség tulajdonságainak felügyelete esetén, a rendszert egy megfelelőbb állapotba lehet juttatni.

Ahelyett, hogy a felügyelet reagálna egy tulajdonság megsértésére, néha lehetséges, hogy megelőzzük a szabálytalanságot események késleltetése vagy megcserélése által, vagy pedig úgy, hogy egyes események bekövetkezését megakadályozzuk. A futási idejű ellenőrzés kutatásának ez a kiterjesztése futási idejű kényszerítésként ismert. Egy kényszerítő felügyelet hatásköre meghatározza a kényszeríthető tulajdonságok osztályát. Események blokkolása elegendő a biztonsági tulajdonságok kényszerítéséhez, ugyanakkor események késleltetésének a képessége lehetővé teszi élőségi tulajdonságok kezelését is.

A FUTÁSI IDEJŰ ELLENŐRZÉS JÖVŐJE

A futási idejű ellenőrzéshez kapcsolódó kutatási területek sokszínűsége bizonyítja téma felkapottságát és a gyakorlati problémák változatosságát, melyeket érint. Az alakuló kutatási területek, mint pl. modell alapú tesztelés, formális verifikáció, hibakeresés, nagy számához való kapcsolódás azt sejteti, hogy a futási idejű ellenőrzés megmarad egy szükséges, kiegészítő technikának anélkül, hogy bármely más területnek alárendelődne.

IRODALOM

Oleg Sokolsky, Klaus Havelund, Insup Lee: „Introduction to the special section on runtime verification”, International Journal on Software Tools for Technology Transfer (2011/13).