Transacties in While

Mehdi Aqadjani Memar, Joost Kraaijeveld, Louis Onrust

Inhoudsopgave

Inleiding

Een While programma zoals beschreven in het boek [Nielson and Nielson, 1999] is in essentie een lineair uitvoerbare sequentie van atomaire statements.

In dit stuk wordt een uitbreiding van While beschreven waarbij arbitraire verzamelingen van statements kunnen worden gebundeld tot atomaire eenheden (transacties) die bovendien parallel aan elkaar kunnen worden uitgevoerd (parallellisme).

Beide concepten kunnen individueel gebruikt worden binnen een programma; interessant wordt het pas wanneer er gekeken wordt naar de mogelijkheden die een combinatie van transacties en parallellisme samen teweeg kunnen brengen.

De opbouw van het werk wordt opgedeeld in drie verschillende stukken:

1. In het eerste stuk worden de uitbreidingen behandeld die nodig zijn
2. In het tweede stuk wordt de formalisatie besproken van de uitbreidingen
3. In het laatste stuk wordt vervolgens een programma besproken en de scenario's die voor kunnen komen worden besproken aan de hand van bewijzen in de vorm van afleidingen

Deel A
Uitbreidingen

In het werk zal voortdurend gebruik gemaakt worden van het begrip programma. Een programma in essentie niets anders dan een verzameling van functies (zie A 2). Deze functies zijn op hun beurt weer een sequentie van statements.

Een programma in While is ook een serie statements, maar hier worden verder geen eisen aan gesteld; $\mathcal{S}[\![z := x]\!](x \mapsto 5)$ kan ook al een geldig programma zijn. Bij ons is een programma hetgeen de programmeur opschrijft (zie C 1). Dus programma als begrip omvat alles wat minimaal geëist wordt door de specificaties, en dit omvat ook het begrip programma uit While, de sequentie van statements. In deze programma's mag gebruik gemaakt worden van alle mogelijkheden die While biedt, maar nu zijn er ook een aantal uitbreidingen die gebruikt mogen worden zoals het opstarten van parallelle taken en het opstarten van een transactionele omgeving met alle voor- en nadelen die ermee verbonden zijn.

A 1 Executieomgeving

Een programma staat meestal niet op zich. In de meeste gevallen wordt een programma opgestart binnen een besturingssysteem. Een dergelijke omgeving zorgt ervoor dat allerlei infrastructuur niet door ieder programma zelf geïmplementeerd hoeft te worden. In een aantal gevallen zullen we omwille van abstractie gebruik maken van een dergelijke omgeving, hier executieomgeving genoemd.

Het is de taak van de executieomgeving om het programma op te starten en alle benodigde variabelen te initialiseren. Verder zorgt de executieomgeving voor het beheer van de taken en zorgt ervoor dat communicatie tussen de taken mogelijk is.

A 1.1 Globaliteit

De executieomgeving is de meest globale omgeving. Dat wilt zeggen dat de executieomgeving alles kan overzien wat er gebeurt in het programma; het heeft dus zicht op alle taken, variabelen en de algemene progressie.

Omdat onze taal het niet toestaat dat een taak variabelen aanspreekt of wijzigt buiten de omgeving van de taak, levert dit in gevallen wanneer de ene taak bijvoorbeeld een resultaat terug wilt geven aan een

andere taak, problemen op. Om dit toch goed te laten verlopen wordt er voor de interactie tussen de twee taken gebruik gemaakt van de executieomgeving als tussenpersoon, zodat de overdracht van gegevens niet alleen goed verloopt, zonder de regels te overtreden, maar ook om deze overdracht als atomaire stap te beschouwen.

Een andere belangrijke taak van de executieomgeving is het in de gaten houden van de taken. Zeker in het geval van interactie tussen taken (zoals die in transacties bijvoorbeeld plaats zullen gaan vinden) is het erg handig om te weten of de taak met wie je wilt samenwerken überhaupt nog wel bestaat. De executieomgeving zal niet per se publiekelijk maken dat een taak gestorven is, maar wanneer er naar gevraagd wordt kan er onder alle omstandigheden een antwoord verkregen worden betreffende de status van de taak.

In sommige gevallen maakt de executieomgeving ook bepaalde dingen publiekelijk, dit geldt voor de zogenaamde labels. Op elk moment is voor elke taak bekend welke functies er allemaal bestaan, maar ook welke andere taken er zijn en welke transacties er allemaal bezig zijn. Labels die expliciet lokaal zijn binnen de taak zijn de variabelen, dit onder andere om te verzekeren dat data-integriteit gegarandeerd kan worden (met als uitzondering de executieomgeving, die deze variabelen wel kan inzien).

Het is voor de individuele taken binnen een programma niet mogelijk om onderling direct te communiceren. Eventuele communicatie (zoals het wachten op elkaar en het doorgeven van resultaten etc) gebeurt door dit aan de executieomgeving "te vragen" door middel van functieaanroepen/systemcalls. Alle communicatie tussen de executieomgeving en individuele taken gebeurt atomair: er is nooit sprake van enig raceconditie als gevolg van deze communicatie.

De state van de executieomgeving is vergelijkbaar met een environment (zoals in Proc). We kennen hier geen speciale naam aan toe, maar we gaan er vanuit dat deze omgeving er gewoon is. Op die manier kunnen we er ook voor zorgen dat taken niet zomaar globale waarden kunnen aanpassen, en dus kan er voor gezorgd worden dat de globale omgeving schoon blijft van onnodige variabelen. We houden de globale omgeving niet expliciet bij, maar waar nodig komt het in de semantiek tot uiting dat er iets aangepast wordt.

A 1.2 Ongedefinieerde variabelen

Een variabele die `undef` is, kan alleen gebruikt worden om te testen of om deze variabel een waarde te geven. Alle andere bewerkingen met een dergelijk variabele leidt tot een abort van het gehele programma.

De executieomgeving houdt bij welke variabelen en welke labels er op een gegeven moment bestaan. De executieomgeving kan dus in het geval van een verwijzing naar een label of variabele die niet (meer) bestaat er voor zorgen dat het programma ophoudt.

Hier zijn ook andere keuzen te maken, maar die zijn niet per se makkelijker als het gaat over het redeneren over het verloop en de uitkomst van een programma:

Ongedefinieerd gedrag Dit is iets wat in de praktijk makkelijker is dan in een formele, speciaal om te bewijzen, opgezette taal. Dit gedrag is overigens niet handig en zeer onwenselijk, aangezien het programma na zo een ongeldige aanroep elke uitkomst kan geven. Dit omvat uitkomsten die per definitie fout zijn, maar ook antwoorden die wel eens goed zouden kunnen zijn. Kortom, aan het antwoord is geen waarde toe te kennen, aangezien deze onder ongedefinieerd gedrag tot stand gekomen is

Exceptie Een andere methode om dit soort problemen op te vangen is het gooien van een exceptie. Met behulp van een exceptie weet je dat er iets mis gegaan is, en kan er naar gehandeld worden.
Dit is een zeer vruchtbaar alternatief voor onze gekozen oplossing; deze uitbreiding vergt echter nog meer uitbreidingen op While bovenop de al door ons voorgestelde uitbreidingen

A 2 Functies

We breiden While uit met de notie van functies. Functies zijn vergelijkbaar met de procedures binnen While zoals beschreven in [Nielson and Nielson, 1999, p. 52]. De definitie van functies is analoog aan die van procedures in While. Bij het opstarten van een programma zal de executieomgeving alle functielabels globaal zichtbaar maken. De volgorde waarin de functies worden gedefinieerd is niet van belang: voordat er een functie wordt uitgevoerd zijn alle functies zichtbaar.

De executieomgeving zal bij het opstarten van een While programma de taak *main* opstarten met als startfunctie de functie met het functielabel *main* en een lege state.

Een programma wordt beëindigd als het einde van de *main*-functie wordt bereikt of als geen enkele taak meer loopt.

Om functies te kunnen voorzien van een naam, breiden we de meta-variabelen en syntactische categorieën uit:

fl geeft een functielabel aan, **FunctionLabel**

Een functie wordt gekenmerkt door de functienaam, de argumenten en de statements die de functie uitvoert. De functienaam en de argumenten van de functie worden ook wel functietype of functiedeclaratie genoemd. De statements binnen de functie worden ook wel functiebody of functie-implementatie genoemd.

De functie die aanroept wordt caller genoemd, en de functie die aangeroepen wordt, wordt callee genoemd.

func fl *tl_callee tl_caller vn tid* D_V `is start` S `end`
→ *fl: FunctieLabel, naam van de functie*
→ *tl_callee: TaakLabel van de callee*
→ *tl_caller: TaakLabel van de caller*
→ *vn: Naam van de result variabele in de caller*
→ *tid: TransactionLabel* Geeft aan van welke transactionele omgeving de callee onderdeel uitmaakt. Dit wordt gebruikt door `collect_votes`. Als *tid* `undef` is, dan is de taak geen onderdeel van een transactie
→ D_V : Lijst met functieargumenten
→ *S: Statements van de functie-implementatie*

A 2.1 Functieargumenten

Argumenten worden doorgegeven aan de functies door een lijst met functieargumenten D_V. De exacte lijst van functieargumenten wordt gedefinieerd door de functie zelf.

Een aantal functieargumenten zijn verplicht en staan als zodanig apart in de lijst met argumenten vermeld. Het is aan de programmeur om deze variabelen van een juiste waarde te voorzien.

Alle functieargumenten worden "by value" doorgegeven. Het is mogelijk om in de functiedeclaratie de argument van een default value te voorzien. In dat geval is het niet nodig voor om bij het aanroepen van de functie de waarden van de variabelen mee te geven, maar zullen de variabelen in de functiebody zelf de default waarde hebben.

De argumenten worden "by name" doorgegeven. Het is niet relevant in welke volgorde de variabelen worden gedeclareerd of meegegeven bij aanroep van een functie. Dat betekent wel dat de namen van de mee te geven variabelen uniek moeten zijn binnen de aanroep.

De scope van de namen van de argumenten begint vanaf `spawn` en eindigt bij het uitvoeren van de laatste statement van de functie. Het is mogelijk om in een `spawn` een argument variabele met de naam x te initialiseren met een gelijknamige lokale variabele: $var\ x := x$.

A 3 Parallellisme

Een While programma bestaat uit een verzameling van taken (minimaal 1) die parallel aan elkaar worden uitgevoerd. Elke taak voert een functie uit. Ieder While programma heeft tenminste één functie — met de naam *main* — die door de executie-omgeving als eerste zal worden uitgevoerd. Iedere taak heeft een eigen state. Het is niet mogelijk om direct, zonder tussenkomst van de executieomgeving, te communiceren tussen taken.

Om taken te kunnen voorzien van een naam, breiden we de meta-variabelen en syntactische categorieën uit:

tl geeft een taaklabel aan, **TaskLabel**

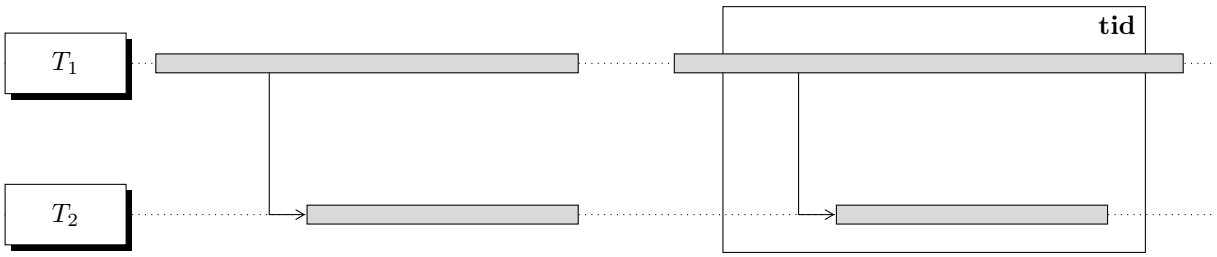
A 3.1 spawn

Om een taak op te starten introduceren we de statement `spawn`; door `spawn` aan te roepen wordt een taak gecreëerd die begint met het uitvoeren van het eerste statement van de functie die meegeven wordt.

De variabelen tl_caller, tl_callee, fl, vn en tid zijn verplichte argumenten. De vraag of D_V als argument meegegeven moet hangt af van de functie met functielabel fl. De waarde van de diverse argumenten is sterk gerelateerd aan de vereisten van de functiedeclaratie van de uit te voeren functie. Wanneer er geen argumenten meegegeven worden, worden de defaultargumenten van de functie gebruikt.

Zoals aangegeven bij `func` krijgt elke taak een variabele tot zijn beschikking in de caller. In `spawn` wordt de naam van deze variabele gezet. Deze variabele is alleen te benaderen met de primitieve functie `set_result`. Totdat `set_result` is aangeroepen voor deze variabele is de waarde van deze variabele `undef`.

Als er sprake is van van een transactionele omgeving alswel een niet-transactionele omgeving, dan moet *tid* een geldige waarde hebben. Als er van een transactionele omgeving geen sprake is, dan krijgt *tid* de waarde `undef`. Als *tid* (transactielabel, zie A 4) gedefinieerd is dan wordt de startstate van de taak opgeslagen.



Een schematische weergave van een `spawn`. Links wordt een taak opgestart zonder gebruik te maken van een transactionele omgeving, rechts wordt er wel gebruik van gemaakt.

spawn *fl tl_callee tl_caller vn tid* D_V
→ *fl: FunctieLabel van de functie*
→ *tl_callee: TaakLabel van de callee*
→ *tl_caller: TaakLabel van de caller*
→ *vn: Naam van de result variabele in de caller*
→ *tid: TransactionLabel* Geeft aan van welke transactionele omgeving de callee onderdeel uitmaakt. Dit wordt gebruikt door `collect_votes`. Als *tid* `undef` is, dan is de taak geen onderdeel van een transactie
→ D_V*: De lijst met functieargumenten*

Start een taak met het meegegeven label. Reserveer variabele *vn* voor de return value van de callee

A 3.2 set_result

Om het resultaat van een taak terug te geven aan de caller introduceren we de (primitieve) booleaanse expressie `set_result`. De callee geeft met behulp van `set_result` de waarde door aan de caller. Omdat de beide taken van elkaar zijn afgeschermd zijn, gebeurt dit door tussenkomst van de executieomgeving.

Omdat niet altijd uit het domein van *waarde* op te maken is, of de berekening goed gegaan is, wordt dit expliciet door de boolean *statusTaak* aangegeven.

Er zijn twee versies van `set_result`. De ene versie is voor gebruik in een niet-transactionele omgeving. De andere is voor gebruik in een transactionele omgeving en heeft een extra argument *tid*.

Als `set_result` wordt aangeroepen in een transactionele omgeving, dan zal `set_result` blokkeren totdat er `commit_transaction` of `rollback_transaction` plaats vindt. Als `set_result` wordt aangeroepen in dezelfde taak als waarin ook `start_transactie` is aangeroepen, dan blokkeert de `set_result` nooit en returnt met dezelfde waarde als de meegegeven statusTaak. Een (potentieel) blokkerende `set_result` retourneert met `true` bij een `commit_transaction` en `false` bij een `rollback_transaction`.

Alle taken die in het uitvoeren van `wait` geblokkeerd zijn, worden onmiddelijk na een `set_result` gedeblokkeerd. De `set_result` en `wait` coördinatie wordt atomair afgehandeld waarbij racecondities worden voorkomen.

Als *statusTaak* `false` is, dan wordt is *a* door de executieomgeving op `undef` gezet, ongeacht de concrete waarde die bij de functieaanroep wordt doorgegeven.

set_result *tl vn a b*
→ *tl: Taaklabel van de caller*
→ *vn: Naam van de result variabele in de caller*
→ *a: Waarde van result variabele*
→ *b: StatusTaak*
← `true` als *b* is true, `false` als *b* is false.

Zet in de caller op de gereserveerde plaats *vn* de waarde *a*

De waarde *b* wordt gebruikt als return value van een eventuele `wait`

Als de caller een `wait` gedaan heeft alvorens de callee een `set_result` gedaan heeft, dan wordt de `wait` van de caller geblokkeerd. Na het uitvoeren van een `set_result` worden eventuele blokkades opgeheven

set_result *tl vn a b tid*
→ *tl: Taaklabel van de caller*
→ *vn: Naam van de result variabele in de caller*
→ *a: Waarde van result variabele*
→ *b: StatusTaak*
→ *tid: TransactionLabel*
← `true` als *tid* en commit. `false` als *tid* en rollback.

Zet in de caller op de gereserveerde plaats *vn* de waarde *a*

De waarde *b* wordt gebruikt als return value van een eventuele `wait`

Als de caller een `wait` gedaan heeft alvorens de callee een `set_result` gedaan heeft, dan wordt de `wait` van de caller geblokkeerd. Na het uitvoeren van een `set_result` worden eventuele blokkades opgeheven

A 3.3 wait

Om te wachten op het resultaat van een taak introduceren we de booleanse expressie `wait`; met behulp van `wait` wacht de caller op het uitvoeren van `set_result` door de callee.

`wait` blokkeert zolang `set_result` niet aangeroepen is door de callee. Als de callee `set_result` heeft aangeroepen voordat `wait` wordt aangeroepen, zal `wait` niet blokkeren. De executieomgeving zal er voor zorgen dat hierbij geen racecondities op kunnen treden.

- Als `set_result` is aangeroepen retourneert `wait` *statusTaak*
- Als de taak wordt beëindigd voordat `set_result` wordt aangeroepen zal de executieomgeving `false` retourneren

Merk op dat de `wait` de enige methode is voor de caller om te achterhalen wat het resultaat van een callee is.

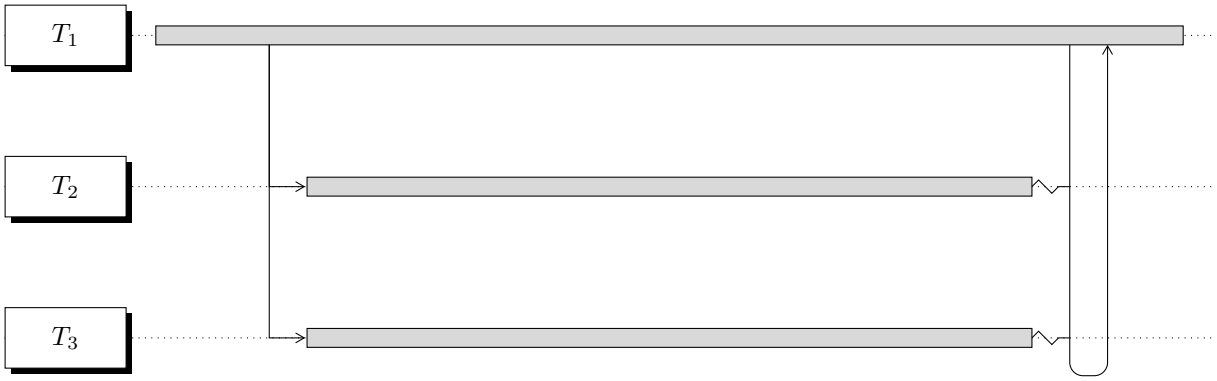
wait *tl*
→ *tl: Taaklabel van de callee*
← `true` als *b* van `set_result true` is. `false` anders.

Een semaphore die de taak — de caller van `wait` — blokkeert totdat er een resultaat bekend is voor de taak *tl*

A 4 Transacties

In While is de atomaire eenheid van executie de enkelvoudige statement.

Wij breiden While uit met de notie van transacties. Transacties zijn arbitraire verzamelingen van statements die gebundeld worden tot atomaire eenheden. Het speciale van deze verzamelingen is dat ze de eigenschap delen dat taken[1] die met elkaar verbonden zijn ofwel allemaal slagen ofwel allemaal niet slagen. Met slagen wordt hier bedoeld dat het programma naar een volgende state gaat, en met niet slagen bedoelen we dat de state van voor de transactie hersteld wordt. In beide gevallen zal het programma verdergaan met de statementsequentie. Dat wil zeggen dat er een afhankelijkheid is tussen de taken die gebaseerd is op het kunnen verkrijgen van resultaten.



Een schematische weergave van het concept parallellisme. Er is een hoofdtaak T_1 die ervoor zorgt dat T_2 en T_3 ontstaan zodat die zelfstandig berekeningen kunnen uitvoeren. Als de taken klaar zijn is er de mogelijkheid om de resultaten te verenigigen.

Transacties voldoen normaliter aan ACID. We leggen de nadruk op ACI, daar waar de durability vooral in praktische situaties erg van belang is zoals beschreven in [Gray, 1981].

Atomair (Atomic) Dit betekent dat als er een onderdeel van de transactie misgaat, dat de hele transactie misgaat. Dit houdt in dat om een transactie te kunnen laten slagen, dat alle taken die

1. Taken kunnen net als het concept van parallellisme individueel voorkomen, dus waar hierboven "taken" genoemd is, kan ook sprake zijn van één taak.

betrekking hebben op de transactie moeten slagen, en wanneer er een taak niet slaagt, alle taken niet slagen.

Het atomaire slaat dus op de transactie als geheel, aangezien die als eenheid moet worden behandeld.

Consistent (Consistency) Consistentie is iets wat afgehandeld kan worden binnen de functies. Omdat While een getypeerde taal is, hebben we sowieso al niet te maken met type-fouten. Wel kunnen we domeinfouten ontstaan, bijvoorbeeld het banksaldo mag niet onder 0 komen; dit moet door de auteur van de functie afgehandeld worden, de taal While, en zelfs het programma als geheel, kennen beiden geen domeinbeperkingen.

Gedurende de berekeningen mogen de beperkingen gelaten worden voor wat ze zijn, maar alvorens het aanroepen van `set_result` is het de taak van de functie om gecontroleerd te hebben of het resultaat überhaupt geldig is.

Geïsoleerd (Isolated) Deze eigenschap hebben we erg duidelijk opgenomen in onze specificaties door ervoor te zorgen dat variabelen en states nooit "by reference" worden meegegeven. Dit zorgt er in essentie voor dat elke taak zijn eigen set met variabelen heeft en dat die veranderd kunnen worden zonder dat dit ook maar enig effect heeft op de variabelen van andere taken.

Het is natuurlijk wel nodig om resultaten bij elkaar te kunnen laten komen, omdat transacties anders beperkt nut hebben. Dit doen wij door gebruik te maken van de executieomgeving die wederom zonder zijeffecten netjes in sommige gevallen variabelen in taken aan kan passen zodat bijvoorbeeld resultaten van de door de caller gespawnde taken in de caller als resultaat geplaatst kunnen worden.

Duurzaam (Durability) Wij houden geen logs bij van welke wijzigingen we wanneer uitgevoerd hebben om welke reden. Dit zou je doorgaans wel willen doen wanneer je waarde hecht aan de duurzaamheid.

Om die reden kunnen we ook niet, wanneer de main-taak een abort krijgt, de taak opnieuw opstarten in dezelfde toestand als waar hij gebleven was; of opstarten in een geldige toestand, en dan vanuit het log opnieuw dezelfde keuzes maken om weer in dezelfde toestand te komen.

Om transactionele omgevingen te kunnen voorzien van een naam, breiden we de meta-variabelen en syntactische categorieën uit:

tid geeft een transactielabel aan, **TransactionLabel**

A 4.1 Transactie

Alle statements binnen een taak en alle taken die gespawnd worden binnen deze taak vormen deel van deze transactie. De transactie wordt gestart met `start_transaction`. De transactie wordt afgesloten met een `commit_transaction` of een `rollback_transaction`. De taak die een transactie start wordt de transactiemanager van de transactie genoemd.

De transactie-omgeving is de verzameling van states en statements van de taken die onderdeel uitmaken van de transactie. Deze omgeving is bekend onder zijn transactielabel, *tid*. De transactielabels zijn globaal bekend.

start_transaction *tid*
    S′
`if collect_votes` *tid*
`then`
    `if` *b*
    `then`
        S″
        `commit_transaction` *tid*
    `else`
        S‴
        `rollback_transaction` *tid*
`else`
    S⁗
    `rollback_transaction` *tid*

Omwille van de overzichtelijkheid wordt deze functie opgedeeld in vier stukken.

A 4.1.1 start_transaction

`start_transaction` markeert het begin van een transactie.

De enige voorwaarde om een transactie op te kunnen starten is dat het een uniek transactielabel krijgt en het label mag niet `undef` zijn.

Bij `start_transaction` wordt een kopie gemaakt van de huidige state, en deze kopie wordt opgeslagen in de state. En het transactielabel wordt toegevoegd aan de set met transactielabels.

start_transaction *tid*
→ *tid: Transactielabel*

A 4.1.2 collect_votes

Verzamelt van alle uitstaande subtaken binnen de transactie het resultaat en als alle resultaten `true` zijn, geeft `collect_votes` `true` weer, anders `false`.

Als `collect_votes` wordt aangeroepen voor taken die nog geen `set_result` hebben aangeroepen dan krijgt `collect_votes` de waarde `false`.

collect_votes *tid*
→ *tid: Transactielabel*
← Levert een `true` op in het geval dat alle statusTaken van alle taken die binnen deze transactie (*tid*) opgestart zijn, `true` zijn, `false` anders

A 4.1.3 commit_transaction

Alle wijzigingen die binnen de transactie in de state van het programma gemaakt zijn worden nu definitief gemaakt en worden behouden. De transactielabel wordt verwijderd en daarmee verdwijnt de transactieomgeving.

`commit_transaction` deblokkeert alle `set_results` van alle onderliggende deelnemende taken met `true`. De functie blokkeert totdat alle eventuele subtaken die onderdeel van de transactie zijn `end_commit_transaction` hebben aangroepen ten teken van het feit dat zijn alle statements die noodzakelijk zijn om de commit van de transactie voor die taak definitief te maken zijn uitgevoerd.

commit_transaction *tid*
→ *tid: Transactielabel*

A 4.1.4 rollback_transaction

De state van een taak gaat na een rollback terug naar de opgeslagen state. De statementsequentie gaat niet terug naar dat punt, maar gaat verder. De transactielabel wordt verwijderd en daarmee verdwijnt de transactieomgeving.

Deblokkeert alle `set_results` van alle onderliggende deelnemende taken met `false`. Indien dit het geval is wordt de huidige state gereset naar een eerder opgeslagen state zoals opgeslagen is door de huidige taak met `start_transactie`. De functie blokkeert totdat alle eventuele subtaken die onderdeel van de transactie zijn `end_rollback_transaction` hebben aangroepen ten teken van het feit dat zijn alle statements die noodzakelijk zijn om de rollback van de transactie voor die taak definitief te maken zijn uitgevoerd.

rollback_transaction *tid*
→ *tid: Transactielabel*

A 4.2 Einde van een transactie door een taak

Als een transactiemanager een `commit_transaction` of `rollback_transaction` uitvoert dan wordt de `set_result` van de onderliggende taken gedeblokkeerd, waarna deze de statements behorend bij `true` of `false` uitvoeren. Zodra de onderliggende taak daarmee gereed is maakt deze dit via de executieomgeving met behulp van `end_commit_transaction` (in het geval van `true`) of `end_commit_transaction` (in het geval van `false`) bekend.

Zodra alle onderliggende taken het einde van de transactie bekend hebben gesteld zal de `commit_transaction` of `rollback_transaction` gedeblokkeerd worden.

end_commit/rollback_transaction
`if set_result` *tl vn a b tid* `then`
    S_1 `end_commit_transaction` *tl tid*
`else`
    S_2 `end_rollback_transaction` *tl tid*

De taak die de transactie gestart is wordt via de executeomgeving op de hoogte gesteld van het feit dat deze taak alle statements met betrekking tot een commit of rollback heeft uitgevoerd

Omwille van de overzichtelijkheid wordt deze statement opgedeeld in twee stukken

A 4.2.1 end_commit_transaction

Met behulp van `end_commit_transaction` maakt een taak bekend dat alle statements met betrekking tot een commit zijn uitgevoerd.

end_commit_transaction *tl tid*
→ *tl: Taaklabel*
→ *tid: Transactielabel*

A 4.2.2 end_rollback_transaction

Met behulp van `end_rollback_transaction` maakt een taak bekend dat alle statements met betrekking tot een rollback zijn uitgevoerd.

end_rollback_transaction *tl tid*
→ *tl: Taaklabel*
→ *tid: Transactielabel*

A 4.3 Individuele transactieafhandeling

De taken zijn verantwoordelijk voor het uitvoeren van een rollback wanneer daar door de caller om gevraagd wordt.

Hoewel in kleine While-programma zonder zijeffecten het nut niet direct zichtbaar is, is er toch voor gekozen om de individuele taken verantwoordelijk te stellen voor het verzorgen van hun eigen commit of rollback.

In een echt programma is de caller doorgaans niet op de hoogte van de implementatie van de callee, en daardoor kan de caller niets zeggen over de acties die ondernomen moeten worden door de callee bij een commit of een rollback.

Deel B

Formalisatie

B 1 Syntax

We nemen de syntax van While over en breiden deze uit:

$$
\begin{array}{rcl}
a & ::= & n \mid x \mid a_1 + a_2 \mid a_1 \times a_2 \mid a_1 - a_2 \mid \texttt{undef} \\
b & ::= & \texttt{true} \mid \texttt{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \wedge b_2 \\
 & \mid & \texttt{wait}\ tl \\
 & \mid & \texttt{set\_result}\ tl\ vn\ a\ b\ tid \\
 & \mid & \texttt{set\_result}\ tl\ vn\ a\ b \\
 & \mid & \texttt{collect\_votes}\ tid \\
S & ::= & x := a \mid \texttt{skip} \mid S_1\ ;\ S_2 \mid \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2 \mid \texttt{while}\ b\ \texttt{do}\ S \\
 & \mid & D_V \\
 & \mid & D_F \\
 & \mid & \texttt{spawn}\ fl\ tl\_callee\ tl\_caller\ vn\ tid\ D_V \\
 & \mid & \texttt{call}\ fl \\
 & \mid & \texttt{start\_transaction}\ tid\ S \\
 & & \texttt{if collect\_votes}\ tid\ \texttt{then} \\
 & & \quad \texttt{if}\ b\ \texttt{then} \\
 & & \quad\quad S_1\ \texttt{commit\_transaction}\ tid \\
 & & \quad \texttt{else} \\
 & & \quad\quad S_2\ \texttt{rollback\_transaction}\ tid \\
 & & \texttt{else} \\
 & & \quad S_3\ \texttt{rollback\_transaction}\ tid \\
 & \mid & \texttt{if set\_result}\ tl\ vn\ a\ b\ tid\ \texttt{then} \\
 & & \quad S_1\ \texttt{end\_commit\_transaction}\ tl\ tid \\
 & & \texttt{else} \\
 & & \quad S_2\ \texttt{end\_rollback\_transaction}\ tl\ tid \\
D_F & ::= & \texttt{func}\ fl\ tl\_callee\ tl\_caller\ vn\ D_V\ tid\ \texttt{is start}\ S\ \texttt{end}\ ;\ D_F \mid \epsilon \\
D_V & ::= & \texttt{var}\ x := a\ ;\ D_V \mid \epsilon
\end{array}
$$

Zoals gesteld in de hoofdstukken A 2, A 3 en A 4 breiden we de meta-variabelen en syntactische categorieën uit:

n	geeft een numeral aan, **Num**
x	geeft een variabele aan, **Var**
a	geeft een aritmetische expressie aan, **Aexp**
b	geeft een booleaanse expressie aan, **Bexp**
S	geeft een statement aan, **Stm**
fl	geeft een functielabel aan, **FunctionLabel**
tl	geeft een taaklabel aan, **TaskLabel**
tid	geeft een transactielabel aan, **TransactionLabel**
vn	geeft een variabelenaam aan, **VariableName**

Deze meta-variabelen mogen zowel met accenten gebruikt worden alswel met subscripts.

B 2 Semantiek

B 2.1 Parallelliteitsoperator

Om het concept van parallelle taken in While te kunnen introduceren moet er een symbool bedacht worden om dit aan te geven. Er is gekozen om gebruik te maken van $\|$. Dit symbool geeft aan dat er parallelle paden van executie plaatsvinden. Anders dan in de While-uitbreiding met parallelisme werken taken echt parallel zonder de notie van interleaving.

Het programma gaat verder met de state die ontstaan is door `spawn`.

$$\gamma \parallel \gamma'$$

γ is de configuratie waarmee het programma verder gaat, dit is dus altijd een state. γ' is van de vorm $\langle S,\ s\rangle$; γ' is dus de parallelle taak.

B 2.2 Undef

`undef` is een speciale waarde die aangeeft dat de waarde van de variabele buiten het domein. Zo is bijvoorbeeld de waarde van de returnvalue van een functie nog niet bekend, terwijl de variabele al wel gedeclareerd moet zijn.

`undef` kan gebruikt voor de volgende situaties:

- Variabelen die `undef` zijn kunnen voor niets anders gebruikt worden dan het testen op het zijn van `undef` en het voorzien van een waarde
- Elk ander gebruik van een variabele die `undef` is leidt tot een abort

B 2.3 Uitbreiding van booleans

In de syntax zijn een viertal booleaanse expressies toegevoegd: `wait`, `set_result` (met en zonder *tid*) en `collect_votes`. Hoewel deze expressies sterke gelijkenissen vertonen met statements of functies, is er voor gekozen om ze van het type booleaanse expressie te maken omdat ze niets wijzigen aan de state van de caller.

Omdat de state van een taak opzichzelf er niet toe doet voor de waarde die `set_result`, `wait` of `collect_votes` toegewezen krijgt, moet er een semantiek zijn die de semantiek uit While uitbreidt, maar wel een constructie toestaat om als het ware te abstraheren van de state die op moment dat de functie wordt aangeroepen geldt voor die taak. Dit hebben we opgelost door een functie $\mathcal{B}'$ te introduceren. Deze functie $\mathcal{B}'$ is alleen gedefinieerd voor de drie eerdergenoemde functies.

Voor de standaardmanier van het converteren naar een boolean wordt de functie $\mathcal{B}$ gebruikt, en voor de drie functies levert deze een aanroep aan naar de nieuwe functie $\mathcal{B}'$ aan met hetzelfde argument:

$$\mathcal{B}[\![\texttt{set\_result}\ tl\ fl\ x\ a\ b]\!]s = \begin{cases} \texttt{tt} & \text{als } \mathcal{B}'[\![\texttt{set\_result}\ tl\ fl\ x\ a\ b]\!] = \texttt{true} \\ \texttt{ff} & \text{als } \mathcal{B}'[\![\texttt{set\_result}\ tl\ fl\ x\ a\ b]\!] = \texttt{false} \end{cases}$$

$$\mathcal{B}[\![\texttt{set\_result}\ tl\ fl\ x\ a\ b\ tid]\!]s = \begin{cases} \texttt{tt} & \text{als } \mathcal{B}'[\![\texttt{set\_result}\ tl\ fl\ x\ a\ b\ tid]\!] = \texttt{true} \\ \texttt{ff} & \text{als } \mathcal{B}'[\![\texttt{set\_result}\ tl\ fl\ x\ a\ b\ tid]\!] = \texttt{false} \end{cases}$$

$$\mathcal{B}[\![\texttt{wait}\ tl]\!]s = \begin{cases} \texttt{tt} & \text{als } \mathcal{B}'[\![\texttt{wait}\ tl]\!] = \texttt{true} \\ \texttt{ff} & \text{als } \mathcal{B}'[\![\texttt{wait}\ tl]\!] = \texttt{false} \end{cases}$$

$$\mathcal{B}[\![\texttt{collect\_votes}\ tid]\!]s = \begin{cases} \texttt{tt} & \text{als } \mathcal{B}'[\![\texttt{collect\_votes}\ tid]\!] = \texttt{true} \\ \texttt{ff} & \text{als } \mathcal{B}'[\![\texttt{collect\_votes}\ tid]\!] = \texttt{false} \end{cases}$$

De executieomgeving zal nu, aan de hand van alles wat bij de executieomgeving bekend is, een waarde toekennen aan de functie zodanig dat:

$$\mathcal{B}'[\![\texttt{set\_result}\ tl\ fl\ x\ a\ b]\!] = \begin{cases} \texttt{true} & \text{als } b = \texttt{true} \\ \texttt{false} & \text{als } b = \texttt{false} \end{cases}$$

$$\mathcal{B}'[\![\texttt{set\_result}\ tl\ fl\ x\ a\ b\ tid]\!] = \begin{cases} \texttt{true} & \text{als commit\_transaction is aangeroepen} \\ \texttt{false} & \text{als rollback\_transaction is aangeroepen} \\ \texttt{true} & \text{anders} \end{cases}$$

$$\mathcal{B}'[\![\texttt{wait}\ tl]\!] = \begin{cases} \texttt{true} & \text{als set\_result van } tl \text{ gezet is met } \texttt{true} \\ \texttt{false} & \text{anders} \end{cases}$$

$$\mathcal{B}'[\![\texttt{collect\_votes}\ tid]\!] = \begin{cases} \texttt{true} & \text{als set\_result van alle } tl \text{ binnen } tid \text{ gezet is met } \texttt{true} \\ \texttt{false} & \text{anders} \end{cases}$$

B 2.4 Hulpfuncties

Op divers plaatsen maken we gebruik van lokale en globale lijsten van variabelen. Om deze lijsten te kunnen administreren maken we gebruik van diverse hulp functies om elementen toe te voegen, op te vragen en te verwijderen.

B 2.4.1 FunctieLabel

GLOBFL(FL) is de globale lijst van functies hun (default) D_V en bijbehorende statements.

Om een functie toe te voegen aan de globale lijst met functies gebruiken we de functie `GLOBFL`.

De semantiek van de functie is:

$$[\texttt{GLOBFL}] \qquad \langle \texttt{GLOBFL}\ \texttt{func}\ fl\ D_V\ S\ ;\ D_F,\ s\rangle \Rightarrow s$$

Het toevoegen van een functie gaat als volgt:

$$\begin{aligned} \texttt{GLOBFL}(\texttt{func}\ fl\ D_V\ S\ ;\ D_F) &= \{\{fl, D_V, S\}\} \cup \text{GLOBFL}(FL) \\ \texttt{GLOBFL}(\epsilon) &= \text{GLOBFL}(FL) \end{aligned}$$

B 2.4.2 TaskLabel

Er worden twee soorten lijsten van taaklabels bijgehouden. De executieomgeving houdt een globale lijst van alle taaklabels bij terwijl een taak een lokale lijst met gespawnde taken bijhoudt.

GLOBTL(TL) is de globale lijst van taaklabels.

Om een taaklabel toe te voegen aan de globale lijst met taaklabels van alle taken gebruiken we de functie `GLOBTL`.

De semantiek van de functie is:

$$[\texttt{GLOBTL}] \qquad \langle \texttt{GLOBTL}\ tl,\ s\rangle \Rightarrow s$$

Het toevoegen gaat als volgt:

$$\texttt{GLOBTL}(tl) = \{tl\} \cup \text{GLOBTL}(TL)$$

Om een taaklabel te verwijderen uit de globale lijst met taaklabels van alle taken gebruiken we de functie `GLOBTL'`.

De semantiek van de functie is:

$$[\texttt{GLOBTL'}] \qquad \langle \texttt{GLOBTL'}\ tl,\ s\rangle \Rightarrow s$$

Het verwijderen gaat als volgt:

$$\texttt{GLOBTL'}(tl) \quad = \quad GLOBTL(TL) \setminus \{tl\}$$

LOCTL(TL) is de lokale lijst van gespawnde taken.

Om een taaklabel toe te voegen aan de lokale lijst met taaklabels met gespawnde taken gebruiken we de functie `LOCTL`.

De semantiek van de functie is:

$$[\texttt{LOCTL}] \qquad \langle \texttt{LOCTL}\ tl,\ s\rangle \Rightarrow s'$$

Het toevoegen gaat als volgt:

$$\texttt{LOCTL}(tl)\ s \quad = \quad \{tl\} \cup \text{LOCTL}(TL)$$

Om een taaklabel te verwijderen uit de lokale lijst met taaklabels met gespawnde taken gebruiken we de functie `LOCTL'`.

De semantiek van de functie is:

$$[\texttt{LOCTL'}] \qquad \langle \texttt{LOCTL'}\ tl,\ s\rangle \Rightarrow s'$$

Het verwijderen gaat als volgt:

$$\texttt{LOCTL'}(tl)\ s \quad = \quad LOCTL(TL) \setminus \{tl\}$$

B 2.4.3 TransactionLabel

Er worden twee soorten lijsten van transactielabels bijgehouden. De executieomgeving houdt een globale lijst van alle transactielabels bij terwijl een taak een lokale lijst met transactielabels van de taak zelf bijhoudt.

GLOBTID(TID) is de globale lijst van transactielabels.

Om de transactielabel toe te voegen aan de globale lijst met transactielabel gebruiken we de functie `GLOBTID`.

De semantiek van de functie is:

$$[\texttt{GLOBTID}] \qquad \langle \texttt{GLOBTID}\ tid,\ s\rangle \Rightarrow s$$

Het toevoegen gaat als volgt:

$$\texttt{GLOBTID}(tid) \quad = \quad \{tid\} \cup \text{GLOBTID}(TID)$$

Om de transactielabel te verwijderen uit de globale lijst met transactielabel gebruiken we de functie `GLOBTID'`.

De semantiek van de functie is:

$$[\texttt{GLOBTID'}] \qquad \langle \texttt{GLOBTID'}\ tid,\ s\rangle \Rightarrow s$$

Het verwijderen gaat als volgt:

$$\texttt{GLOBTID'}(tid) \quad = \quad \texttt{GLOBTID}(TID) \setminus \{tid\}$$

LOCTID(TID) is de lokale lijst van transactielabels.

Om de transactielabel toe te voegen aan de lokale lijst met transactielabel gebruiken we de functie `LOCTID`.

De semantiek van de functie is:

$$[\texttt{LOCTID}] \qquad \langle \texttt{LOCTID}\ tid,\ s\rangle \Rightarrow s'$$

Het toevoegen gaat als volgt:

$$\texttt{LOCTID}(tid) \quad = \quad \{tid\} \cup \text{LOCTID}(TID)$$

Om de transactielabel te verwijderen uit de lokale lijst met transactielabel gebruiken we de functie `LOCTID'`.

De semantiek van de functie is:

$$[\texttt{LOCTID'}] \qquad \langle \texttt{LOCTID'}\ tid,\ s\rangle \Rightarrow s'$$

Het verwijderen gaat als volgt:

$$\texttt{LOCTID'}(tid) \quad = \quad \text{LOCTID}(TID) \setminus \{tid\}$$

B 2.4.4 State

GLOBSTATE($STATE$) is de globale lijst van states.

Om een state toe te voegen aan de globale lijst met states gebruiken we de functie `GLOBSTATE`.

De semantiek van de functie is:

$$[\texttt{GLOBSTATE}] \qquad \langle \texttt{GLOBSTATE}\ tl\ tid\ s',\ s\rangle \Rightarrow s$$

Het toevoegen gaat als volgt:

$$\texttt{GLOBSTATE}(tl\ tid,\ s) \quad = \quad \{tl, tid, s\} \cup \text{GLOBSTATE}(STATE)$$

Om een state op te vragen uit de globale lijst met states gebruiken we de functie `GLOBSTATE''`.

De semantiek van de functie is:

$$[\texttt{GLOBSTATE''}] \qquad \langle \texttt{GLOBSTATE''}\ tl\ tid\rangle \Rightarrow s$$

Het opvragen gaat als volgt:

$$\texttt{GLOBSTATE''}(tl\ tid) \quad = \quad \{tl, tid, s\}$$

Om een state toe te verwijderen uit de globale lijst met states gebruiken we de functie `GLOBSTATE'`.

De semantiek van de functie is:

$$[\texttt{GLOBSTATE'}] \qquad \langle \texttt{GLOBSTATE'}\ tl\ tid,\ s\rangle \Rightarrow s$$

Het verwijderen gaat als volgt:

$$\texttt{GLOBSTATE'}(tl\ tid) \quad = \quad \texttt{GLOBSTATE}(STATE) \setminus \{tl, tid, s\} \quad \text{Waar } s = \texttt{GLOBSTATE''}\ (tl\ tid)$$

B 2.5 Statements

We nemen de bestaande statements uit While over en voegen daar onze statements aan toe. Vanuit While nemen we over:

$$[\texttt{ass}] \quad \langle x := a.\ s\rangle \Rightarrow s[x \mapsto \mathcal{A}[\![a]\!]s]$$

$$[\texttt{skip}] \quad \langle \texttt{skip},\ s\rangle \Rightarrow s$$

$$[\texttt{comp}^1] \quad \frac{\langle S_1,\ s\rangle \Rightarrow \langle S_1',\ s'\rangle}{\langle S_1\ ;\ S_2,\ s\rangle \Rightarrow \langle S_1'\ S_2,\ s'\rangle}$$

$$[\texttt{comp}^2] \quad \frac{\langle S_1,\ s\rangle \Rightarrow s'}{\langle S_1\ ;\ S_2,\ s\rangle \Rightarrow \langle S_2,\ s'\rangle}$$

$$[\texttt{if}^{\texttt{true}}] \quad \langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2,\ s\rangle \Rightarrow \langle S_1,\ s\rangle \ \text{ Als } \mathcal{B}[\![b]\!]s = \texttt{tt}$$

$$[\texttt{if}^{\texttt{false}}] \quad \langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2,\ s\rangle \Rightarrow \langle S_2,\ s\rangle \ \text{ Als } \mathcal{B}[\![b]\!]s = \texttt{ff}$$

$$[\texttt{while}] \quad \langle \texttt{while}\ b\ \texttt{do}\ S,\ s\rangle \Rightarrow \langle \texttt{if}\ b\ \texttt{then}\ (S\ ;\ \texttt{while}\ b\ \texttt{do}\ S)\ \texttt{else}\ \texttt{skip},\ s\rangle$$

En we voegen daar de statements uit de volgende paragrafen aan toe.

B 2.5.1 D_V en var

D_V is een lijst met variabelen, waarbij de variabelen gescheiden worden door een ";". De lijst wordt afgesloten door middel van een ϵ (de lege variabele).

$$[\texttt{var}^1] \quad \langle \texttt{var}\ x := a\ ;\ D_V,\ s[x \mapsto \mathcal{A}[\![a]\!]s]\rangle \Rightarrow \langle D_V,\ s'\rangle$$

$$[\texttt{var}^2] \quad \langle \epsilon,\ s\rangle \Rightarrow s$$

Wanneer we $\texttt{var}\ x := a$ aanroepen, dan kijken we of x al bestaat in de huidige state:

- Wanneer x nog niet bestaat, dan wordt er een variabele x geïntroduceerd met de waarde a
- Wanneer x al bestaat, dan wordt de x uit de `var` in scope gebracht. Dat betekend dat de $s\,x = y$ een andere x, zeg x_1, is dan de x, zeg x_2, uit $s'\,x = z$. Deze x_2 verdwijnt wanneer de functie uit scope gaat, zeg in state s''. Na het uit scope gaan van x_2 krijgen we weer $s''\,x_1 = y$

B 2.5.2 D_F en func

Zoals gesteld in A 2 wordt bij het opstarten van het programma de functie *main* uitgevoerd en worden alle functielabels globaal gemaakt. Om dit te kunnen doen moeten de functies ingelezen door middel van de functie `func`.

$[\texttt{func}^1]$ zorgt ervoor dat de functie gelezen wordt, en de functielabel globaal samen met zijn defaultargumente worden toegevoegd aan de set met functielabels. $[\texttt{func}^2]$ zorgt ervoor dat als er geen functiedeclaraties meer gevonden worden er met de *main* begonnen wordt. En deze functie $[\texttt{func}^2]$ lost dat op.

$$[\texttt{func}^1] \quad \langle \texttt{func}\ fl\ tl\_callee\ tl\_caller\ vn\ D_V\ \texttt{is start}\ S\ \texttt{end}\ ;\ D_F,\ s\rangle \Rightarrow s$$

Waar we toepassen $\texttt{GLOBFL}(\texttt{func}\ fl\ D_V\ S\ ;\ D_F)$

$$[\texttt{func}^2] \quad \langle \epsilon,\ s\rangle \Rightarrow \langle \texttt{spawn}\ fl := main\ tl\_callee := main\ tl\_caller := \texttt{undef}\ vn := \texttt{undef}\ tid = \texttt{undef},\ s\rangle$$

B 2.5.3 call

FUNC_F haalt door middel van een aanroep aan de executieomgeving de statements op die de functieimplementatie vormen, van de functie die globaal bekend is onder de naam van het argument, de functielabel.

$$\text{FUNC}_\text{F} = \text{FunctionLabel} \hookrightarrow \text{Stm}$$

$$[\texttt{call}] \quad \langle \texttt{call}\ fl,\ s\rangle \Rightarrow \langle S,\ s\rangle \ \text{ Waar } \text{FUNC}_\text{F}\ fl = \{S, D_V\}$$

B 2.5.4 spawn

[spawn] $\langle \texttt{spawn}\ fl\ tl\_callee\ tl\_caller\ vn\ tid\ D_V,\ s\rangle \Rightarrow s$
$\| \langle D'_V\ ;\ D_V\ ;\ \texttt{call}\ fl,\ s[\texttt{tl} \mapsto \mathcal{T}[\![tl\_callee]\!]s][\texttt{tid} \mapsto \mathcal{TL}[\![tid]\!]s][\texttt{vn} \mapsto \mathcal{VN}[\![vn]\!]s]\rangle$

Waar we toepassen GLOBTL(tl) en LOCTL(tl) s op γ van $\gamma \| \gamma'$ en waar $\text{FUNC}_\text{F}\ fl = \{S, D'_V\}$

vn moet de naam zijn van een lokale variabele in de caller

B 2.5.5 transaction

In het begin van transactie wordt de state waarmee de transactie begint gekopieerd. Dit gebeurt bij het treffen van het woord `start_transaction`.

[transaction] $\langle$`start_transaction` $tid\ S$
`if collect_votes` tid `then`
S' `if` b `then`
S'' `commit_transaction` tid
`else`
S''' `rollback_transaction` tid
`else`
S'''' `rollback_transaction` tid
$, s\rangle \Rightarrow$
$\langle S\ ;$ `if collect_votes` tid `then`
S' `if` b `then`
S'' `commit_transaction` tid
`else`
S''' `rollback_transaction` tid
`else`
S'''' `rollback_transaction` tid
$, s[\texttt{tid} \mapsto \mathcal{TL}[\![tid]\!]\rangle$

Waar we toepassen GLOBTID(tid) en GLOBSTATE($tl\ tid$)

B 2.5.6 commit

[commit] $\langle$`commit_transaction` $tid,\ s\rangle \Rightarrow s$

Waar we toepassen GLOBTID(tid)

B 2.5.7 rollback

[rollback] $\langle$`rollback_transaction` $tid,\ s\rangle \Rightarrow s'$

s' is waarde verkregen van GLOBSTATE'' met tid en tl uit s. Waar we toepassen GLOBTID'(tid)

B 2.5.8 end_transaction

Als een subtaak met behulp van set_result tid het resultaat wil zetten

$$[\texttt{end\_transaction}^{\texttt{tt}}] \quad \langle \texttt{if set\_result } tl\ vn\ a\ b\ tid \texttt{ then } S_1 \texttt{ end\_commit\_transaction } tl\ tid \texttt{ else } S_2 \texttt{ end\_rollback\_transaction } tl\ tid, s\rangle \Rightarrow \langle S_1 \texttt{ end\_commit\_transaction } tl\ tid, s\rangle$$

$$\left[\texttt{end\_transaction}^{\texttt{ff}}\right] \quad \langle \texttt{if set\_result } tl\ vn\ a\ b\ tid \texttt{ then } S_1 \texttt{ end\_commit\_transaction } tl\ tid \texttt{ else } S_2 \texttt{ end\_rollback\_transaction } tl\ tid, s\rangle \Rightarrow \langle S_2 \texttt{ end\_rollback\_transaction } tl\ tid, s\rangle$$

B 2.5.9 end_commit

$$[\texttt{end\_commit}] \quad \langle \texttt{end\_commit\_transaction } tl\ tid,\ s\rangle \Rightarrow s$$

Waar we toepassen GLOBTID'(tid) en LOCTID'(tid)

B 2.5.10 end_rollback

$$[\texttt{end\_rollback}] \quad \langle \texttt{end\_rollback\_transaction } tl\ tid,\ s\rangle \Rightarrow s'$$

s' is waarde verkregen van GLOBSTATE'' met tid en tl uit s. Waar we toepassen GLOBTID'(tid) en LOCTID'(tid)

Deel C

Programma

C 1 Programma

Met behulp van een concreet programma gaan we drie scenario's bekijken:

1. Alles gaat goed
2. De berekening levert een domeinfout op
3. Er vindt een abort plaats

We doen dat met onderstaand programma waarvan we zelf makkelijk de uitkomst kunnen bepalen, en dus ook kunnen controleren of het programma hetzelfde resultaat oplevert. De resultaten die we verwachten respectievelijk aan de scenario's hierboven:

1. Taak1 levert 3 op, taak2 levert 7 op, en totale resultaat is 10
2. Taak2 wordt opgestart met de waarden $x := 5$ en $y := 5$, hierdoor kan het programma niet een geldig antwoord geven en moet er dus een rollback plaatsvinden
3. Een van de twee taken sterft af door onbekende oorzaak; ook hiervan is het netto resultaat dat er een rollback plaats gaat vinden

Functie 1: Optellen

```
func optellen tl tl vn tid var x:=2; var y:=3; ϵ is
start
  result := x + y;
  if result < 10
  then
    valid := true
  else
    valid := false
  if set_result tl vn result valid tid then
    skip end_commit_transaction tl tid
  else
    skip end_rollback_transaction tl tid
end
```

Functie 2: Main

```
func main tl tl vn tid ϵ is
start
  start_transaction transactie1
  main_1:=undef;
  main_2:=undef;
  spawn fl:=optellen tl_callee:=taak1 tl_caller:=main vn:=main_1 tid:=transactie1
    var x:=1; var y:=2; ϵ;
  spawn fl:=optellen tl_callee:=taak2 tl_caller:=main vn:=main_2 tid:=transactie1
    var x:=3; var y:=4; ϵ;
  if wait taak1 then skip else skip;
  if wait taak2 then skip else skip
  if collect_votes transactie1 then
    grand_total := main_1 + main_2;
    if grand_total < 15 then
      skip;
      commit_transaction transactie1
    else
      skip;
      rollback_transaction transactie1
  else
    grand_total := 0;
    rollback_transaction transactie1
end
```

C 2 Bewijs

De bewijzen zullen vanwege de omvang gepresenteerd worden in twee vormen. Allereerst zal in dit hoofdstuk de korte versie vermeld staan. Deze korte versie laat niet steeds alle statements zien, maar toont wel de functienaam van de functie die gebruikt wordt, en toont de huidige state op dat moment. Met behulp van deze versie is het mogelijk het grote bewijs na te gaan, of zelf het bewijs te volgen wanneer de statements allemaal uitgeschreven worden.

De tweede versie is de complete versie, dus compleet met functienamen, states en de complete statementsequentie. Omdat de grootte van deze versies weinig gering zijn, zijn ze als los bestand meegeleverd. Dit bestand is te bezichtigen op `http://trac.askesis.nl/tidbits/browser/SeC/aux/Sec-Transactie-Bewijs`.

`ods` of op `http://trac.askesis.nl/tidbits/browser/SeC/aux/Sec-Transactie-Bewijs.xls` of via de bijlage bij dit document.

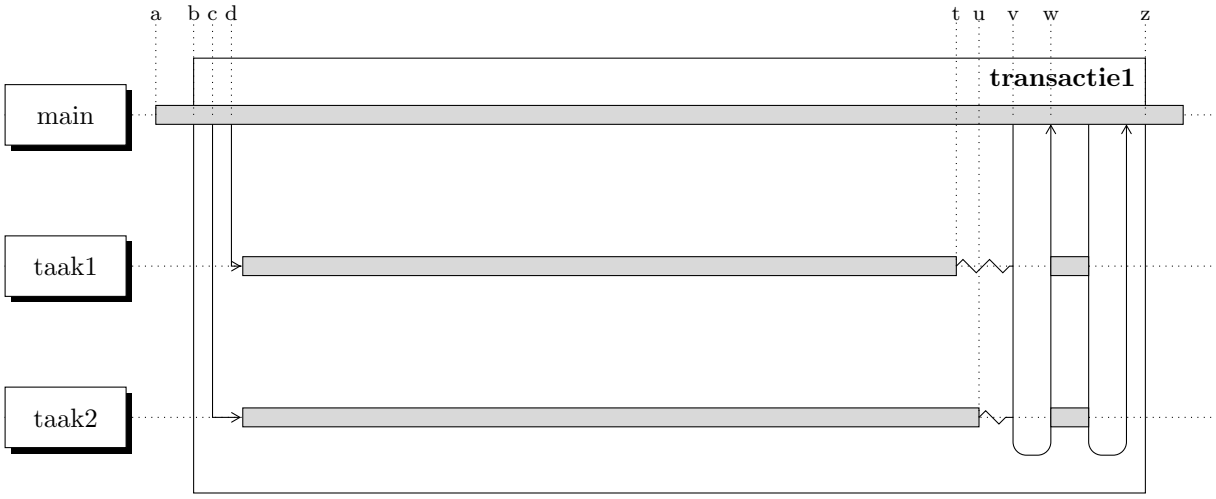
Elke bewijsvoering wordt begeleid door een afbeelding die een schets van de opbouw van het programma weergeeft. In deze afbeeldingen zijn markers en in het bewijs zullen deze markers ook verschijnen zodat de lezer zich beter kan oriënteren en het duidelijker is in welke context een bepaalde state zich bevindt.

Het bewijs dient van boven naar beneden gelezen te worden, behalve de taken op zichzelf, die mogen parallel bekeken worden (we kennen toch geen racecondities). De state aangegeven is de state die geldt op het moment dat je de functie besluit te kiezen. Het resultaat van die functie staat dus één regel lager. Voor een complete bewijsboom, zie bijlage.

C 2.1 Commit

Bij een commit willen we graag bewijzen dat beide taken een antwoord teruggeven aan de hoofdtaak, en dat deze antwoorden verkregen zijn op een manier waarvan de taak dacht dat het goed was.

In dit bewijs worden twee taken gespawned uit *main* die de taken hebben om 1 bij 2 op te tellen, en om 3 bij 4 op te tellen. We verwachten dus dat *main* respectievelijk de waarden 3 en 7 binnenkrijgt. Als *main* deze waarden vervolgens optelt moet aan het einde van het programma `grand_total` de waarde 10 hebben.



Dit is het programma zoals we gebruiken voor het bewijs:

Functie 3: Commit

```
func optellen tl tl vn tid var x:=2; var y:=3; ϵ is
start
  result := x + y;
  if result < 10
  then
    valid := true
  else
    valid := false
  if set_result tl vn result valid tid then
    skip end_commit_transaction tl tid
  else
    skip end_rollback_transaction tl tid
end;
func main tl tl vn tid ϵ is
start
  start_transaction transactie1
```

```
  main_1:=undef;
  main_2:=undef;
  spawn fl:=optellen tl_callee:=taak1 tl_caller:=main vn:=main_1 tid:=transactie1
    var x:=1; var y:=2; ϵ;
  spawn fl:=optellen tl_callee:=taak2 tl_caller:=main vn:=main_2 tid:=transactie1
    var x:=3; var y:=4; ϵ;
  if wait taak1 then skip else skip;
  if wait taak2 then skip else skip
  if collect_votes transactie1 then
    grand_total := main_1 + main_2;
    if grand_total < 15 then
      skip;
      commit_transaction transactie1
    else
      skip;
      rollback_transaction transactie1
  else
    grand_total := 0;
    rollback_transaction transactie1
end;
ϵ
```

		Executieomgeving
	`func1`	s
	`func1`	s
	`func2`	s
	`spawn`	s
		s

		main
a	`var2`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \texttt{undef}][\texttt{vn} \mapsto \texttt{undef}]$
	`var2`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \texttt{undef}][\texttt{vn} \mapsto \texttt{undef}]$
	`call`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \texttt{undef}][\texttt{vn} \mapsto \texttt{undef}]$
b	`transaction`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \texttt{undef}][\texttt{vn} \mapsto \texttt{undef}]$
	`ass`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \mathit{transactie}1][\texttt{vn} \mapsto \texttt{undef}]$
	`ass`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \mathit{transactie}1][\texttt{vn} \mapsto \texttt{undef}][\texttt{main\_1} \mapsto \texttt{undef}]$
c	`spawn`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \mathit{transactie}1][\texttt{vn} \mapsto \texttt{undef}][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$
d	`spawn`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \mathit{transactie}1][\texttt{vn} \mapsto \texttt{undef}][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$
	`if tt`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \mathit{transactie}1][\texttt{vn} \mapsto \texttt{undef}][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$
	`skip`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \mathit{transactie}1][\texttt{vn} \mapsto \texttt{undef}][\texttt{main\_1} \mapsto 3][\texttt{main\_2} \mapsto \texttt{undef}]$
	`if tt`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \mathit{transactie}1][\texttt{vn} \mapsto \texttt{undef}][\texttt{main\_1} \mapsto 3][\texttt{main\_2} \mapsto \texttt{undef}]$
	`skip`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \mathit{transactie}1][\texttt{vn} \mapsto \texttt{undef}][\texttt{main\_1} \mapsto 3][\texttt{main\_2} \mapsto 7]$
v	`if tt`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \mathit{transactie}1][\texttt{vn} \mapsto \texttt{undef}][\texttt{main\_1} \mapsto 3][\texttt{main\_2} \mapsto 7]$
w	`ass`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \mathit{transactie}1][\texttt{vn} \mapsto \texttt{undef}][\texttt{main\_1} \mapsto 3][\texttt{main\_2} \mapsto 7]$
	`if tt`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \mathit{transactie}1][\texttt{vn} \mapsto \texttt{undef}][\texttt{main\_1} \mapsto 3][\texttt{main\_2} \mapsto 7]$ $[\texttt{grand\_total} \mapsto 10]$
	`skip`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \mathit{transactie}1][\texttt{vn} \mapsto \texttt{undef}][\texttt{main\_1} \mapsto 3][\texttt{main\_2} \mapsto 7]$ $[\texttt{grand\_total} \mapsto 10]$
z	`commit`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \mathit{transactie}1][\texttt{vn} \mapsto \texttt{undef}][\texttt{main\_1} \mapsto 3][\texttt{main\_2} \mapsto 7]$ $[\texttt{grand\_total} \mapsto 10]$
		$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \mathit{transactie}1][\texttt{vn} \mapsto \texttt{undef}][\texttt{main\_1} \mapsto 3][\texttt{main\_2} \mapsto 7]$ $[\texttt{grand\_total} \mapsto 10]$

		$\mathit{taak}1$
	`var1`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \mathit{transactie}1][\texttt{vn} \mapsto \mathit{main}_1][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto \mathit{main}][\texttt{tl\_callee} \mapsto \mathit{taak}1]$
	`var1`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \mathit{transactie}1][\texttt{vn} \mapsto \mathit{main}_1][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto \mathit{main}][\texttt{tl\_callee} \mapsto \mathit{taak}1][\texttt{x} \mapsto 2]$
	`var2`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \mathit{transactie}1][\texttt{vn} \mapsto \mathit{main}_1][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto \mathit{main}][\texttt{tl\_callee} \mapsto \mathit{taak}1][\texttt{x} \mapsto 2][\texttt{y} \mapsto 3]$
	`var1`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \mathit{transactie}1][\texttt{vn} \mapsto \mathit{main}_1][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto \mathit{main}][\texttt{tl\_callee} \mapsto \mathit{taak}1][\texttt{x} \mapsto 2][\texttt{y} \mapsto 3]$
	`var1`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \mathit{transactie}1][\texttt{vn} \mapsto \mathit{main}_1][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto \mathit{main}][\texttt{tl\_callee} \mapsto \mathit{taak}1][\texttt{x} \mapsto 1][\texttt{y} \mapsto 3]$
	`var2`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \mathit{transactie}1][\texttt{vn} \mapsto \mathit{main}_1][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto \mathit{main}][\texttt{tl\_callee} \mapsto \mathit{taak}1][\texttt{x} \mapsto 1][\texttt{y} \mapsto 2]$
	`call`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \mathit{transactie}1][\texttt{vn} \mapsto \mathit{main}_1][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto \mathit{main}][\texttt{tl\_callee} \mapsto \mathit{taak}1][\texttt{x} \mapsto 1][\texttt{y} \mapsto 2]$
	`ass`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \mathit{transactie}1][\texttt{vn} \mapsto \mathit{main}_1][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto \mathit{main}][\texttt{tl\_callee} \mapsto \mathit{taak}1][\texttt{x} \mapsto 1][\texttt{y} \mapsto 2]$
	`if tt`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \mathit{transactie}1][\texttt{vn} \mapsto \mathit{main}_1][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto \mathit{main}][\texttt{tl\_callee} \mapsto \mathit{taak}1][\texttt{x} \mapsto 1][\texttt{y} \mapsto 2][\texttt{result} \mapsto 3]$
	`ass`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \mathit{transactie}1][\texttt{vn} \mapsto \mathit{main}_1][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto \mathit{main}][\texttt{tl\_callee} \mapsto \mathit{taak}1][\texttt{x} \mapsto 1][\texttt{y} \mapsto 2][\texttt{result} \mapsto 3]$
t	`if tt`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \mathit{transactie}1][\texttt{vn} \mapsto \mathit{main}_1][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto \mathit{main}][\texttt{tl\_callee} \mapsto \mathit{taak}1][\texttt{x} \mapsto 1][\texttt{y} \mapsto 2][\texttt{result} \mapsto 3][\texttt{valid} \mapsto \texttt{true}]$

end trans tt	s[tl ↦ undef][tid ↦ *transactie*1][vn ↦ $main_1$][main_1 ↦ undef][main_2 ↦ undef]
	[tl_caller ↦ *main*][tl_callee ↦ *taak*1][x ↦ 1][y ↦ 2][result ↦ 3][valid ↦ true]
skip	s[tl ↦ undef][tid ↦ *transactie*1][vn ↦ $main_1$][main_1 ↦ undef][main_2 ↦ undef]
	[tl_caller ↦ *main*][tl_callee ↦ *taak*1][x ↦ 1][y ↦ 2][result ↦ 3][valid ↦ true]
	s[tl ↦ undef][tid ↦ *transactie*1][vn ↦ $main_1$][main_1 ↦ undef][main_2 ↦ undef]
	[tl_caller ↦ *main*][tl_callee ↦ *taak*1][x ↦ 1][y ↦ 2][result ↦ 3][valid ↦ true]

	*taak*2
var1	s[tl ↦ undef][tid ↦ *transactie*1][vn ↦ $main_2$][main_1 ↦ undef][main_2 ↦ undef]
	[tl_caller ↦ *main*][tl_callee ↦ *taak*2]
var1	s[tl ↦ undef][tid ↦ *transactie*1][vn ↦ $main_2$][main_1 ↦ undef][main_2 ↦ undef]
	[tl_caller ↦ *main*][tl_callee ↦ *taak*2][x ↦ 2]
var2	s[tl ↦ undef][tid ↦ *transactie*1][vn ↦ $main_2$][main_1 ↦ undef][main_2 ↦ undef]
	[tl_caller ↦ *main*][tl_callee ↦ *taak*2[x ↦ 2][y ↦ 3]
var1	s[tl ↦ undef][tid ↦ *transactie*1][vn ↦ $main_2$][main_1 ↦ undef][main_2 ↦ undef]
	[tl_caller ↦ *main*][tl_callee ↦ *taak*2][x ↦ 2][y ↦ 3]
var1	s[tl ↦ undef][tid ↦ *transactie*1][vn ↦ $main_2$][main_1 ↦ undef][main_2 ↦ undef]
	[tl_caller ↦ *main*][tl_callee ↦ *taak*2][x ↦ 3][y ↦ 3]
var2	s[tl ↦ undef][tid ↦ *transactie*1][vn ↦ $main_2$][main_1 ↦ undef][main_2 ↦ undef]
	[tl_caller ↦ *main*][tl_callee ↦ *taak*2][x ↦ 3][y ↦ 4]
call	s[tl ↦ undef][tid ↦ *transactie*1][vn ↦ $main_2$][main_1 ↦ undef][main_2 ↦ undef]
	[tl_caller ↦ *main*][tl_callee ↦ *taak*2][x ↦ 3][y ↦ 4]
ass	s[tl ↦ undef][tid ↦ *transactie*1][vn ↦ $main_2$][main_1 ↦ undef][main_2 ↦ undef]
	[tl_caller ↦ *main*][tl_callee ↦ *taak*2][x ↦ 3][y ↦ 4]
if tt	s[tl ↦ undef][tid ↦ *transactie*1][vn ↦ $main_2$][main_1 ↦ undef][main_2 ↦ undef]
	[tl_caller ↦ *main*][tl_callee ↦ *taak*2][x ↦ 3][y ↦ 4][result ↦ 7]
ass	s[tl ↦ undef][tid ↦ *transactie*1][vn ↦ $main_2$][main_1 ↦ undef][main_2 ↦ undef]
	[tl_caller ↦ *main*][tl_callee ↦ *taak*2][x ↦ 3][y ↦ 4][result ↦ 7]
if tt	s[tl ↦ undef][tid ↦ *transactie*1][vn ↦ $main_2$][main_1 ↦ undef][main_2 ↦ undef] ——— u
	[tl_caller ↦ *main*][tl_callee ↦ *taak*2][x ↦ 3][y ↦ 4][result ↦ 7][valid ↦ true]
end trans tt	s[tl ↦ undef][tid ↦ *transactie*1][vn ↦ $main_2$][main_1 ↦ undef][main_2 ↦ undef]
	[tl_caller ↦ *main*][tl_callee ↦ *taak*2][x ↦ 3][y ↦ 4][result ↦ 7][valid ↦ true]
skip	s[tl ↦ undef][tid ↦ *transactie*1][vn ↦ $main_2$][main_1 ↦ undef][main_2 ↦ undef]
	[tl_caller ↦ *main*][tl_callee ↦ *taak*2][x ↦ 3][y ↦ 4][result ↦ 7][valid ↦ true]
	s[tl ↦ undef][tid ↦ *transactie*1][vn ↦ $main_2$][main_1 ↦ undef][main_2 ↦ undef]
	[tl_caller ↦ *main*][tl_callee ↦ *taak*2][x ↦ 3][y ↦ 4][result ↦ 7][valid ↦ true]

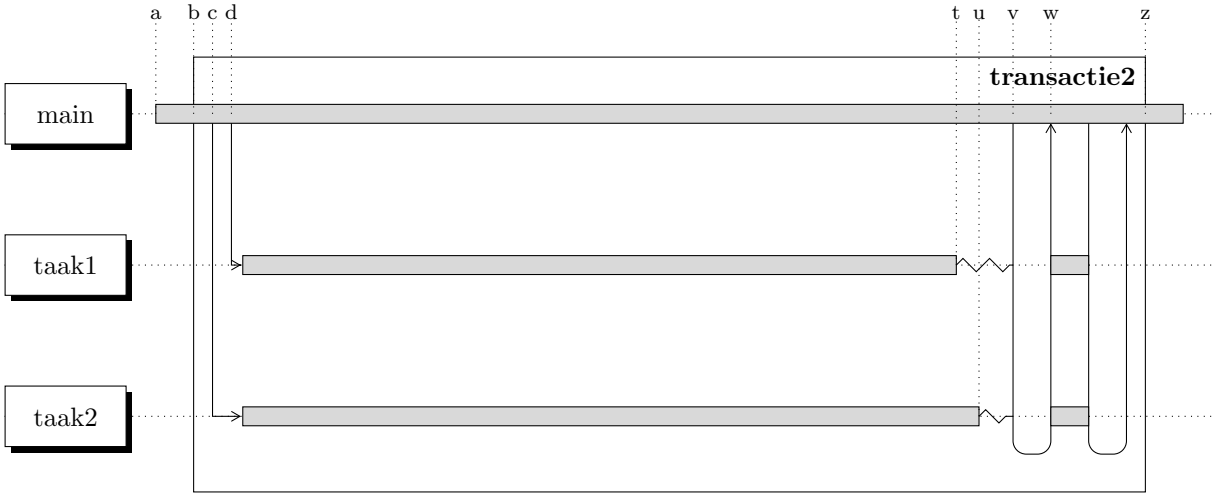
De waarden van `main_1` en `main_2` zijn inderdaad respectievelijk 3 en 7, en `grand_total` heeft inderdaad de waarde 10.

C 2.2 Rollback

Bij een rollback willen we graag bewijzen dat als er een taak is die er niet in slaagt om op juiste wijze tot een antwoord te komen, dat de hele transactie alsware het een atomair blok teruggaat naar de state waarin de transactie gestart is.

In het bewijs zal *taak*1 de waarden 4 en 5 optellen. Omdat deze onder de grens liggen van 10, zal deze taak vinden dat de antwoorden juist zijn en goed genoeg zijn om door te geven aan de caller. *taak*2 daarentegen zal de waarden 5 en 5 optellen. Hiermee komt *taak*2 niet door de domeincontrole die ervoor zorgt dat de som niet boven de 10 mag komen. *taak*2 zegt dat hij niet tot juiste antwoorden heeft kunnen komen, en als reactie daarop zal *main* een rollback in gang zetten.

We verwachten dat de state van *main* aan het einde van het programma gelijk is aan de state die *main* kreeg na het spawnen. Ook de taken die betrokken zijn bij de transactie zullen teruggaan naar de state waarmee ze gespawned zijn.



Functie 4: Rollback

```
func optellen tl tl vn tid var x:=2; var y:=3; ϵ is
start
  result := x + y;
  if result < 10
  then
    valid := true
  else
    valid := false
  if set_result tl vn result valid tid then
    skip end_commit_transaction tl tid
  else
    skip end_rollback_transaction tl tid
end;
func main tl tl vn tid ϵ is
start
  start_transaction transactie2
  main_1:=undef;
  main_2:=undef;
  spawn fl:=optellen tl_callee:=taak1 tl_caller:=main vn:=main_1 tid:=transactie2
    var x:=4; var y:=5; ϵ;
  spawn fl:=optellen tl_callee:=taak2 tl_caller:=main vn:=main_2 tid:=transactie2
    var x:=5; var y:=5; ϵ;
  if wait taak1 then skip else skip;
  if wait taak2 then skip else skip
  if collect_votes transactie2 then
    grand_total := main_1 + main_2;
    if grand_total < 15 then
      skip;
      commit_transaction transactie2
    else
      skip;
      rollback_transaction transactie2
  else
    grand_total := 0;
    rollback_transaction transactie2
end;
ϵ
```

	Executieomgeving	
func1	s	
func1	s	
func2	s	
spawn	s	
	s	

	main	
var2	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \texttt{undef}][\texttt{vn} \mapsto \texttt{undef}]$	a
call	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \texttt{undef}][\texttt{vn} \mapsto \texttt{undef}]$	
transaction	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \texttt{undef}][\texttt{vn} \mapsto \texttt{undef}]$	b
ass	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}2][\texttt{vn} \mapsto \texttt{undef}]$	
ass	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}2][\texttt{vn} \mapsto \texttt{undef}][\texttt{main\_1} \mapsto \texttt{undef}]$	
spawn	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}2][\texttt{vn} \mapsto \texttt{undef}][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$	c
spawn	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}2][\texttt{vn} \mapsto \texttt{undef}][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$	d
if tt	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}2][\texttt{vn} \mapsto \texttt{undef}][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$	
skip	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}2][\texttt{vn} \mapsto \texttt{undef}][\texttt{main\_1} \mapsto 9][\texttt{main\_2} \mapsto \texttt{undef}]$	
if ff	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}2][\texttt{vn} \mapsto \texttt{undef}][\texttt{main\_1} \mapsto 9][\texttt{main\_2} \mapsto \texttt{undef}]$	
skip	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}2][\texttt{vn} \mapsto \texttt{undef}][\texttt{main\_1} \mapsto 9][\texttt{main\_2} \mapsto \texttt{undef}]$	
if ff	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}2][\texttt{vn} \mapsto \texttt{undef}][\texttt{main\_1} \mapsto 9][\texttt{main\_2} \mapsto \texttt{undef}]$	v
ass	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}2][\texttt{vn} \mapsto \texttt{undef}][\texttt{main\_1} \mapsto 9][\texttt{main\_2} \mapsto \texttt{undef}]$	
rollback	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}2][\texttt{vn} \mapsto \texttt{undef}][\texttt{main\_1} \mapsto 9][\texttt{main\_2} \mapsto \texttt{undef}]$	w,z
	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \texttt{undef}][\texttt{vn} \mapsto \texttt{undef}]$	

	$taak1$	
var1	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}2][\texttt{vn} \mapsto \textit{main}_1][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto \textit{main}][\texttt{tl\_callee} \mapsto \textit{taak}1]$	
var1	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}2][\texttt{vn} \mapsto \textit{main}_1][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto \textit{main}][\texttt{tl\_callee} \mapsto \textit{taak}1][\texttt{x} \mapsto 2]$	
var2	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}2][\texttt{vn} \mapsto \textit{main}_1][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto \textit{main}][\texttt{tl\_callee} \mapsto \textit{taak}1][\texttt{x} \mapsto 2][\texttt{y} \mapsto 3]$	
var1	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}2][\texttt{vn} \mapsto \textit{main}_1][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto \textit{main}][\texttt{tl\_callee} \mapsto \textit{taak}1][\texttt{x} \mapsto 2][\texttt{y} \mapsto 3]$	
var1	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}2][\texttt{vn} \mapsto \textit{main}_1][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto \textit{main}][\texttt{tl\_callee} \mapsto \textit{taak}1][\texttt{x} \mapsto 4][\texttt{y} \mapsto 3]$	
var2	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}2][\texttt{vn} \mapsto \textit{main}_1][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto \textit{main}][\texttt{tl\_callee} \mapsto \textit{taak}1][\texttt{x} \mapsto 4][\texttt{y} \mapsto 5]$	
call	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}2][\texttt{vn} \mapsto \textit{main}_1][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto \textit{main}][\texttt{tl\_callee} \mapsto \textit{taak}1][\texttt{x} \mapsto 4][\texttt{y} \mapsto 5]$	
ass	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}2][\texttt{vn} \mapsto \textit{main}_1][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto \textit{main}][\texttt{tl\_callee} \mapsto \textit{taak}1][\texttt{x} \mapsto 4][\texttt{y} \mapsto 5]$	
if tt	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}2][\texttt{vn} \mapsto \textit{main}_1][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto \textit{main}][\texttt{tl\_callee} \mapsto \textit{taak}1][\texttt{x} \mapsto 4][\texttt{y} \mapsto 5][\texttt{result} \mapsto 9]$	
ass	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}2][\texttt{vn} \mapsto \textit{main}_1][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto \textit{main}][\texttt{tl\_callee} \mapsto \textit{taak}1][\texttt{x} \mapsto 4][\texttt{y} \mapsto 5][\texttt{result} \mapsto 9]$	
if ff	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}2][\texttt{vn} \mapsto \textit{main}_1][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto \textit{main}][\texttt{tl\_callee} \mapsto \textit{taak}1][\texttt{x} \mapsto 4][\texttt{y} \mapsto 5][\texttt{result} \mapsto 9]$ $[\texttt{valid} \mapsto \texttt{true}]$	t
skip	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}2][\texttt{vn} \mapsto \textit{main}_1][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto \textit{main}][\texttt{tl\_callee} \mapsto \textit{taak}1][\texttt{x} \mapsto 4][\texttt{y} \mapsto 5][\texttt{result} \mapsto 9]$ $[\texttt{valid} \mapsto \texttt{true}]$	
end_rollback	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}2][\texttt{vn} \mapsto \textit{main}_2][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto \textit{main}][\texttt{tl\_callee} \mapsto \textit{taak}1][\texttt{x} \mapsto 4][\texttt{y} \mapsto 5][\texttt{result} \mapsto 9]$ $[\texttt{valid} \mapsto \texttt{true}]$	

$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto transactie2][\texttt{vn} \mapsto main_2][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$
$[\texttt{tl\_caller} \mapsto main][\texttt{tl\_callee} \mapsto taak1]$

		$taak2$
	var1	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto transactie2][\texttt{vn} \mapsto main_2][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto main][\texttt{tl\_callee} \mapsto taak2]$
	var1	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto transactie2][\texttt{vn} \mapsto main_2][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto main][\texttt{tl\_callee} \mapsto taak2][\texttt{x} \mapsto 2]$
	var2	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto transactie2][\texttt{vn} \mapsto main_2][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto main][\texttt{tl\_callee} \mapsto taak2][\texttt{x} \mapsto 2][\texttt{y} \mapsto 3]$
	var1	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto transactie2][\texttt{vn} \mapsto main_2][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto main][\texttt{tl\_callee} \mapsto taak2][\texttt{x} \mapsto 2][\texttt{y} \mapsto 3]$
	var1	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto transactie2][\texttt{vn} \mapsto main_2][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto main][\texttt{tl\_callee} \mapsto taak2][\texttt{x} \mapsto 5][\texttt{y} \mapsto 3]$
	var2	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto transactie2][\texttt{vn} \mapsto main_2][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto main][\texttt{tl\_callee} \mapsto taak2][\texttt{x} \mapsto 5][\texttt{y} \mapsto 5]$
	call	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto transactie2][\texttt{vn} \mapsto main_2][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto main][\texttt{tl\_callee} \mapsto taak2][\texttt{x} \mapsto 5][\texttt{y} \mapsto 5]$
	ass	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto transactie2][\texttt{vn} \mapsto main_2][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto main][\texttt{tl\_callee} \mapsto taak2][\texttt{x} \mapsto 5][\texttt{y} \mapsto 5]$
	if ff	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto transactie2][\texttt{vn} \mapsto main_2][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto main][\texttt{tl\_callee} \mapsto taak2][\texttt{x} \mapsto 5][\texttt{y} \mapsto 5][\texttt{result} \mapsto 10]$
	ass	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto transactie2][\texttt{vn} \mapsto main_2][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto main][\texttt{tl\_callee} \mapsto taak2][\texttt{x} \mapsto 5][\texttt{y} \mapsto 5][\texttt{result} \mapsto 10]$
u	if ff	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto transactie2][\texttt{vn} \mapsto main_2][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto main][\texttt{tl\_callee} \mapsto taak2][\texttt{x} \mapsto 5][\texttt{y} \mapsto 5][\texttt{result} \mapsto 10]$ $[\texttt{valid} \mapsto \texttt{false}]$
	skip	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto transactie2][\texttt{vn} \mapsto main_2][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto main][\texttt{tl\_callee} \mapsto taak2][\texttt{x} \mapsto 5][\texttt{y} \mapsto 5][\texttt{result} \mapsto 10]$ $[\texttt{valid} \mapsto \texttt{false}]$
	end_rollback	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto transactie2][\texttt{vn} \mapsto main_2][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto main][\texttt{tl\_callee} \mapsto taak2][\texttt{x} \mapsto 5][\texttt{y} \mapsto 5][\texttt{result} \mapsto 10]$ $[\texttt{valid} \mapsto \texttt{false}]$
		$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto transactie2][\texttt{vn} \mapsto main_2][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto main][\texttt{tl\_callee} \mapsto taak2]$

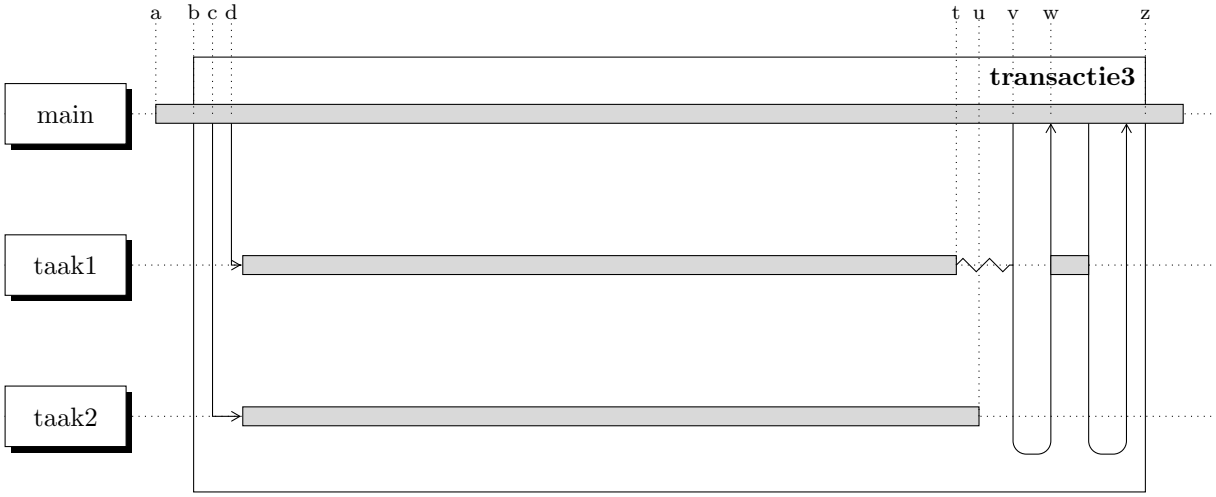
Alle taken, dus *main*, *taak*1 en *taak*2 zijn allemaal terug naar de state waarin ze gespawned zijn omdat er een rollback in gang gezet was. Dit betekent dat het rollbackmechanisme zijn werk gedaan heeft.

C 2.3 Abort

Bij de het bewijzen dat het systeem blijft functioneren als er een abort plaats vindt, gaat het om hoe dit waargenomen wordt, en hoe hier mee omgegaan wordt. Zoals dit werk beschrijft zullen we bij een abort een rollback doen.

Omdat we geen statement voor abort hebben, maar de abort iets is dat van buitenaf komt, is er ook geen speciale notatie voor. De executieomgeving kan wel waarnemen of er een abort optreedt, vandaar dat er op geanticipeerd kan worden.

We verwachten dat als *taak*2 een abort krijgt, alle taken behalve de taken die een abort gehad hebben, teruggaan naar de state zoals die was wanneer ze eenmaal gespawned waren.



Functie 5: Abort

```
func optellen tl tl vn tid var x:=2; var y:=3; ϵ is
start
  result := x + y;
  if result < 10
  then
    valid := true
  else
    valid := false
  if set_result tl vn result valid tid then
    skip end_commit_transaction tl tid
  else
    skip end_rollback_transaction tl tid
end;
func main tl tl vn tid ϵ is
start
  start_transaction transactie3
  main_1:=undef;
  main_2:=undef;
  spawn fl:=optellen tl_callee:=taak1 tl_caller:=main vn:=main_1 tid:=transactie3
    var x:=1; var y:=2; ϵ;
  spawn fl:=optellen tl_callee:=taak2 tl_caller:=main vn:=main_2 tid:=transactie3
    var x:=3; var y:=4; ϵ;
  if wait taak1 then skip else skip;
  if wait taak2 then skip else skip
  if collect_votes transactie3 then
    grand_total := main_1 + main_2;
    if grand_total < 15 then
      skip;
      commit_transaction transactie3
    else
      skip;
      rollback_transaction transactie3
  else
    grand_total := 0;
    rollback_transaction transactie3
end;
ϵ
```

		Executieomgeving
	`func1`	s
	`func1`	s
	`func2`	s
	`spawn`	s
		s

		main
a	`var2`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \texttt{undef}][\texttt{vn} \mapsto \texttt{undef}]$
	`call`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \texttt{undef}][\texttt{vn} \mapsto \texttt{undef}]$
b	`transaction`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \texttt{undef}][\texttt{vn} \mapsto \texttt{undef}]$
	`ass`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}3][\texttt{vn} \mapsto \texttt{undef}]$
	`ass`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}3][\texttt{vn} \mapsto \texttt{undef}][\texttt{main\_1} \mapsto \texttt{undef}]$
c	`spawn`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}3][\texttt{vn} \mapsto \texttt{undef}][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$
d	`spawn`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}3][\texttt{vn} \mapsto \texttt{undef}][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$
	`if tt`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}3][\texttt{vn} \mapsto \texttt{undef}][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$
	`skip`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}3][\texttt{vn} \mapsto \texttt{undef}][\texttt{main\_1} \mapsto 9][\texttt{main\_2} \mapsto \texttt{undef}]$
	`if ff`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}3][\texttt{vn} \mapsto \texttt{undef}][\texttt{main\_1} \mapsto 9][\texttt{main\_2} \mapsto \texttt{undef}]$
	`skip`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}3][\texttt{vn} \mapsto \texttt{undef}][\texttt{main\_1} \mapsto 9][\texttt{main\_2} \mapsto \texttt{undef}]$
v	`if ff`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}3][\texttt{vn} \mapsto \texttt{undef}][\texttt{main\_1} \mapsto 9][\texttt{main\_2} \mapsto \texttt{undef}]$
	`ass`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}3][\texttt{vn} \mapsto \texttt{undef}][\texttt{main\_1} \mapsto 9][\texttt{main\_2} \mapsto \texttt{undef}]$
w,z	`rollback`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}3][\texttt{vn} \mapsto \texttt{undef}][\texttt{main\_1} \mapsto 9][\texttt{main\_2} \mapsto \texttt{undef}]$
		$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \texttt{undef}][\texttt{vn} \mapsto \texttt{undef}]$

		*taak*1
	`var1`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}3][\texttt{vn} \mapsto \textit{main}_1][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto \textit{main}][\texttt{tl\_callee} \mapsto \textit{taak}1]$
	`var1`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}3][\texttt{vn} \mapsto \textit{main}_1][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto \textit{main}][\texttt{tl\_callee} \mapsto \textit{taak}1][\texttt{x} \mapsto 2]$
	`var2`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}3][\texttt{vn} \mapsto \textit{main}_1][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto \textit{main}][\texttt{tl\_callee} \mapsto \textit{taak}1[\texttt{x} \mapsto 2][\texttt{y} \mapsto 3]$
	`var1`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}3][\texttt{vn} \mapsto \textit{main}_1][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto \textit{main}][\texttt{tl\_callee} \mapsto \textit{taak}1]$
	`var1`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}3][\texttt{vn} \mapsto \textit{main}_1][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto \textit{main}][\texttt{tl\_callee} \mapsto \textit{taak}1][\texttt{x} \mapsto 4]$
	`var2`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}3][\texttt{vn} \mapsto \textit{main}_1][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto \textit{main}][\texttt{tl\_callee} \mapsto \textit{taak}1][\texttt{x} \mapsto 4][\texttt{y} \mapsto 5]$
	`call`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}3][\texttt{vn} \mapsto \textit{main}_1][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto \textit{main}][\texttt{tl\_callee} \mapsto \textit{taak}1][\texttt{x} \mapsto 4][\texttt{y} \mapsto 5]$
	`ass`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}3][\texttt{vn} \mapsto \textit{main}_1][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto \textit{main}][\texttt{tl\_callee} \mapsto \textit{taak}1][\texttt{x} \mapsto 4][\texttt{y} \mapsto 5]$
	`if tt`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}3][\texttt{vn} \mapsto \textit{main}_1][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto \textit{main}][\texttt{tl\_callee} \mapsto \textit{taak}1][\texttt{x} \mapsto 4][\texttt{y} \mapsto 5][\texttt{result} \mapsto 9]$
	`ass`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}3][\texttt{vn} \mapsto \textit{main}_1][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto \textit{main}][\texttt{tl\_callee} \mapsto \textit{taak}1][\texttt{x} \mapsto 4][\texttt{y} \mapsto 5][\texttt{result} \mapsto 9]$
t	`if ff`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}3][\texttt{vn} \mapsto \textit{main}_1][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto \textit{main}][\texttt{tl\_callee} \mapsto \textit{taak}1][\texttt{x} \mapsto 4][\texttt{y} \mapsto 5][\texttt{result} \mapsto 9]$ $[\texttt{valid} \mapsto \texttt{true}]$
	`skip`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}3][\texttt{vn} \mapsto \textit{main}_1][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto \textit{main}][\texttt{tl\_callee} \mapsto \textit{taak}1][\texttt{x} \mapsto 4][\texttt{y} \mapsto 5][\texttt{result} \mapsto 9]$ $[\texttt{valid} \mapsto \texttt{true}]$
	`end_rollback`	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto \textit{transactie}3][\texttt{vn} \mapsto \textit{main}_2][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto \textit{main}][\texttt{tl\_callee} \mapsto \textit{taak}1][\texttt{x} \mapsto 4][\texttt{y} \mapsto 5][\texttt{result} \mapsto 9]$ $[\texttt{valid} \mapsto \texttt{true}]$

$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto transactie3][\texttt{vn} \mapsto main_2][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$
$[\texttt{tl\_caller} \mapsto main][\texttt{tl\_callee} \mapsto taak1]$

	$taak2$
var1	$s[\texttt{tl} \mapsto \texttt{undef}][\texttt{tid} \mapsto transactie3][\texttt{vn} \mapsto main_2][\texttt{main\_1} \mapsto \texttt{undef}][\texttt{main\_2} \mapsto \texttt{undef}]$ $[\texttt{tl\_caller} \mapsto main][\texttt{tl\_callee} \mapsto taak2]$ De taak beëindigt om onbekende redenen

$taak1$ en $main$ zijn inderdaad teruggegaan naar de startstate en daarmee is bewezen dat wanneer een abort optreedt in een taak, deze door het het systeem goed afgehandeld wordt.

Conclusie

C 2.4 Bespreking

In het voorgaand hebben we een aantal uitbreidingen aan de taal While toegevoegd. De kern van de uitbreidingen bestaat uit uitbreidingen om transacties in parallelle taken uit te voeren.

Wij hebben er voor gekozen om de uitbredingen minimaal te houden. Dat betekent dat een aantal gevallen die in "echte parallelle transacties" voorkomen niet met behulp van deze uitbreidingen afgehandeld kunnen worden.

Het is nu mogelijk dat het programma nooit uit een `wait` toestand komt: als een taak een `while true skip` uitvoert zonder ooit `set_result` aan te roepen zal een caller die `wait` aanroept voor deze callee nooit uit de blokkerende toestand van `wait` komen.

Het niet ontwaken uit een blokkerende toestand zou verholpen kunnen worden door het invoeren van timers: als een taak langer dan n-tijd geblokkerd is dan returnt de functie met een speciale waarde waarna het programma weer verder kan gaan. Om capaciteitsredenen is een dergelijke uitbreiding buiten beschouwing gelaten.

Het transactiemechanisme, in essentie een 2-phase commit implementatie, is kwetsbaar voor het aborten van de taak die de transactie is begonnen: als die taak tussen de `collect_votes` en de `commit_transaction` abort dan zullen de onderliggende taken niet uit de blokkerende toestand van `set_result` komen. Een vergelijkbaar opmerking kan gemaakt worden over `commit/rollback_transaction` en `end_commit/rollback_transaction`, maar dan voor een abort van een van de onderliggende taken. Dit is een bekend probleem voor 2-phase commit in het algemeen: zie [Skeen and Stonebraker, 1983].

Referenties

[Gray, 1981] Gray, J. (1981). The transaction concept: Virtues and limitations (invited paper). In *Very Large Data Bases, 7th International Conference, September 9-11, 1981, Cannes, France, Proceedings*, pages 144–154. IEEE Computer Society.

[Nielson and Nielson, 1999] Nielson, H. R. and Nielson, F. (1999). *Semantics with Applications: A Formal Introduction*. Wiley.

[Skeen and Stonebraker, 1983] Skeen, D. and Stonebraker, M. (1983). A formal model of crash recovery in a distributed system. *IEEE Transactions on Software Engineering*, 9(3):219–228.